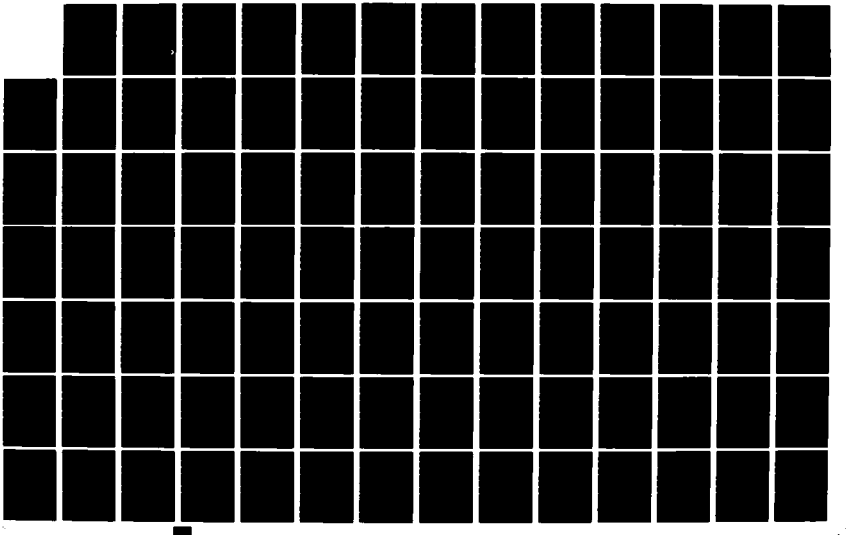
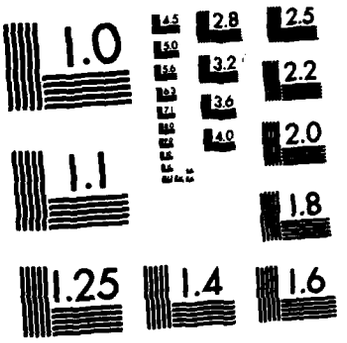
AD-A124 398 A SYNTACTIC APPROACH AND VLSI ARCHITECTURES FOR SEISMIC 1/3
SIGNAL CLASSIFICATION(U) PURDUE UNIV LAFAYETTE IN
SCHOOL OF ELECTRICAL ENGINEERING H LIU ET AL. JAN 83
UNCLASSIFIED N00014-79-C-0574 F/G 8/11 NL





# A SYNTACTIC APPROACH AND VLSI ARCHITECTURES FOR SEISMIC SIGNAL CLASSIFICATION

Hsi-Ho Liu and K. S. Fu
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

January 1983



This work was supported by ONR Contract N00014-79-C-0574 and the NSF-Grant ECS 80-16580.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. AD-A124 398 / 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) A SYNTACTIC APPROACH AND VLSI ARCHITECTURES FOR SEISMIC SIGNAL CLASSIFICATION, | 5. TYPE OF REPORT & PERIOD COVERED Technical / 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Hsi-Ho Liu and K. S. Fu | 8. CONTRACT OR GRANT NUMBER(s) ONR N00014-79-C-0574, |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue University School of Electrical Engineering West Lafayette, IN 47907 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington, VA 22217 | 12. REPORT DATE January 1983 / 13. NUMBER OF PAGES 208 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | 15. SECURITY CLASS. (of this report) unclassified / 15a. DECLASSIFICATION DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release: distribution unlimited.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Syntactic pattern recognition has been applied to seismic classification in this study. Its performance is better than many existing statistical approaches. VLSI architectures for syntactic seismic recognition are also proposed which take advantage of parallel processing and pipelining so that a constant time complexity is attainable when processing large amount of data. Application of syntactic pattern recognition to damage assessment is also proposed and demonstrated on a set of experimental data.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Seismic waveforms are represented by strings of primitives, i.e., sentences, in this study. String-to-string similarity measures based on both distance and likelihood concepts are discussed along with the symmetric property and the hierarchy. A fixed-length segmentation is used in the experiment. Encouraging results comparable to those of the best statistical approaches are obtained with only two very simple features, namely, zero-crossing count and log energy. Primitives are automatically selected using a hierarchical clustering procedure and two decision criteria.

Nearest-neighbor decision rule and finite-state error-correcting parsers are used for classification. For error-correcting parsing, finite-state grammars are first inferred from the training samples. These two approaches have same performance in the experiment, whereas the nearest-neighbor rule is faster in speed.

Attributed grammar and its parsing are also proposed for seismic recognition, which could reduce the complexity and increase the descriptive flexibility of the pattern gramamrs. VLSI architectures are proposed for fast recognition of seismic waveforms. Three systolic arrays perform the feature selection, primitive recognition and string distance computation. These individual units can be used in other similar applications.

Although this study is on seismic classification, it can be extended or modified to tackle other signal recognition problems.




# TABLE OF CONTENTS

Page

# LIST OF TABLES

# LIST OF FIGURES

ABSTRACT

Syntactic pattern recognition has been applied to seismic classification in this study. Its performance is better than many existing statistical approaches. VLSI architectures for syntactic seismic recognition are also proposed which take advantage of parallel processing and pipelining so that a constant time complexity is attainable when processing large amount of data. Application of syntactic pattern recognition to damage assesment is also proposed and demonstrated on a set of experimental data.

Seismic waveforms are represented by strings of primitives, i.e., sentences, in this study. String-to-string similarity mensures based on both distance and likelihood concepts are discussed along with the symmetric property and the hierarchy. A fixed-length segmentation is used in the experiment. Encouraging results comparable to those of the best statistical approaches are obtained with only two very simple features, namely, zero-crossing count and log energy. Primitives are automatically selected using a hierarchical clustering procedure and two decision criteria.

Nearest-neighbor decision rule and finite-state error-correcting parsers are used for classification. For error-correcting parsing, finite-state grammars are first inferred from the training samples. These two approaches have same performance in the experiment, whereas the nearest-neighbor rule is faster in speed.

Attributed grammar and its parsing are also proposed for seismic recognition, which could reduce the complexity and increase the descriptive flexibility of the pattern grammars. VLSI architectures are proposed for fast recognition of seismic waveforms. Three systolic arrays perform the feature selection, primitive recognition and string distance computation. These individual units can be used in other similar applications.

Although this study is on seismic classification, it can be extended or modified to tackle other signal recognition problems.

# CHAPTER I

# INTRODUCTION

## 1.1 Statement of the Problem

In the past, seismic wave analyses were all retained within the geophysical field. Underground structure and earthquake analyses are the most important topics. The major parameters computed from the recorded seismograms are the location, time, depth and magnitude of the event and so forth.

In the 1960's, a new problem arose when the idea of the comprehensive nuclear test ban treaties were proposed. The problem is how to discriminate between the natural earthquake and the secret underground nuclear explosion by seismological methods, which in turn are based on the seismic wave recordings (Bolt, 1976; Dahlman and Israelson, 1977). Traditional methods use the informations like time, location, depth, magnitude, complexity, ratio of body wave magnitude to surface wave magnitude and usually interaction of human experts. However, these methods are not reliable for small events and require the involvement of many seismic stations. Recently, pattern recognition has been applied to the discrimination between these two categories (see Chen, 1978).

It is sometimes very difficult to distinguish between some earthquakes and explosions just by looking at the seismic signals only. Even for experienced analyst additional informations are needed in order to make correct classification. According to the source mechanism, the explosion signal should look more like pulse and contain higher frequency than earthquake, while the earthquake signal should last longer and look more complex. However it is not always true since the depth of the source, distance and geophysical configuration of the path will change the waveform significantly. Here are some examples. The difference between explosion and earthquake is very clear in Figure 1.1, but not so in Figure 1.2 and Figure 1.3 where neither frequency nor complexity can tell the difference. In pattern recognition terminology these two classes are overlapped.

All the existing pattern recognition applications use statistical approach. Since the complexity and structural information play an important role in seismic analysis, it is thus natural to pursue syntactic (structural) approach in seismic pattern analysis. In oil exploration, the structure of the seismic reflection indicates the underground structure. In earthquake / explosion classification, the structural information is the most important feature. The block diagram of a typical syntactic pattern recognition system is shown in Figure 1.4. Due to the unknown characteristic about the source and environment, seismic grammar is usually difficult to construct manually. Therefore, grammatical inference techniques will be applied to infer the pattern grammar from a set of training samples. An error-correcting parser will also be used because the chance that a testing sample is perfectly accepted by the inferred grammar is very slim. This is usually a rule rather than

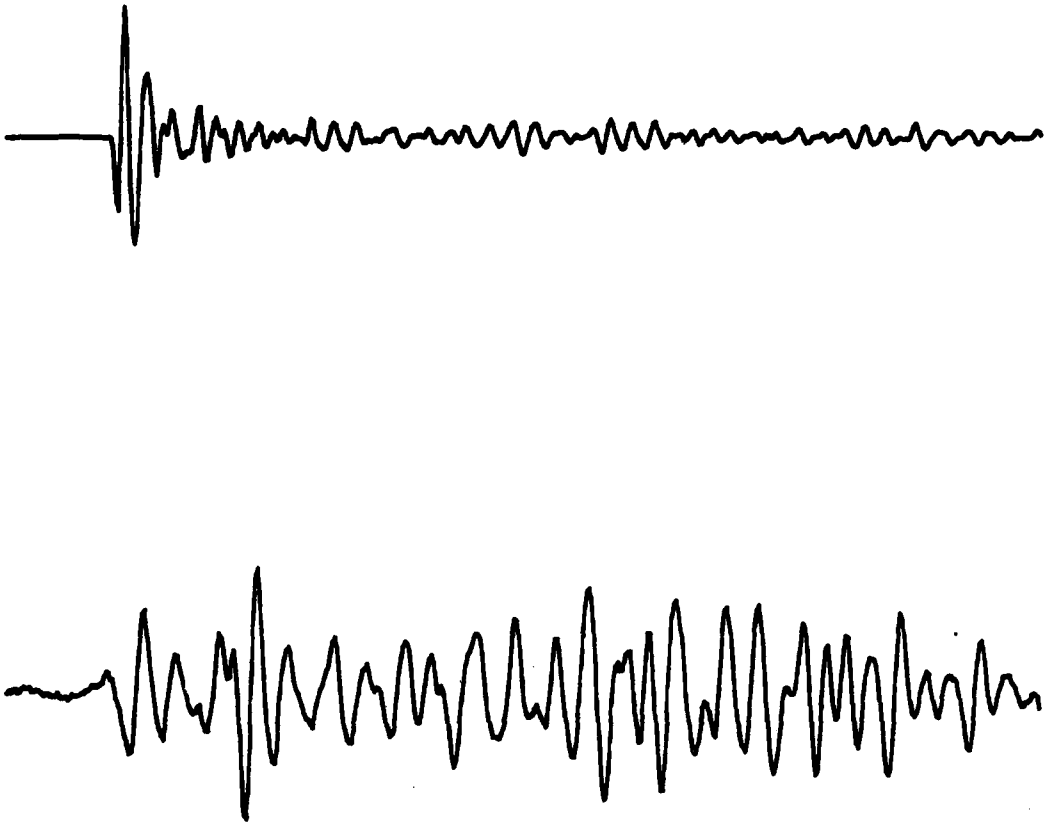


Figure 1.1 An example of two typical seismic records. The top one is an explosion; the bottom one is an earthquake.

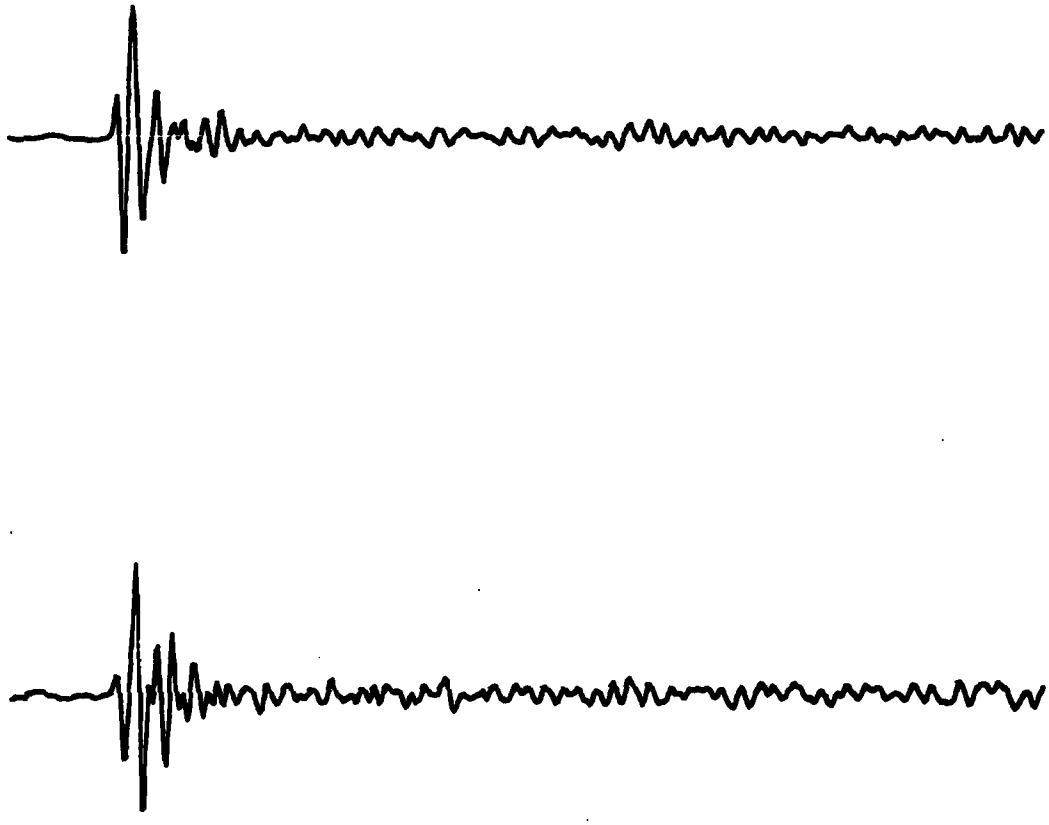


Figure 1.2 An example of extreme case. The top one is a typical explosion waveform; the bottom one is an earthquake record which looks like an explosion.

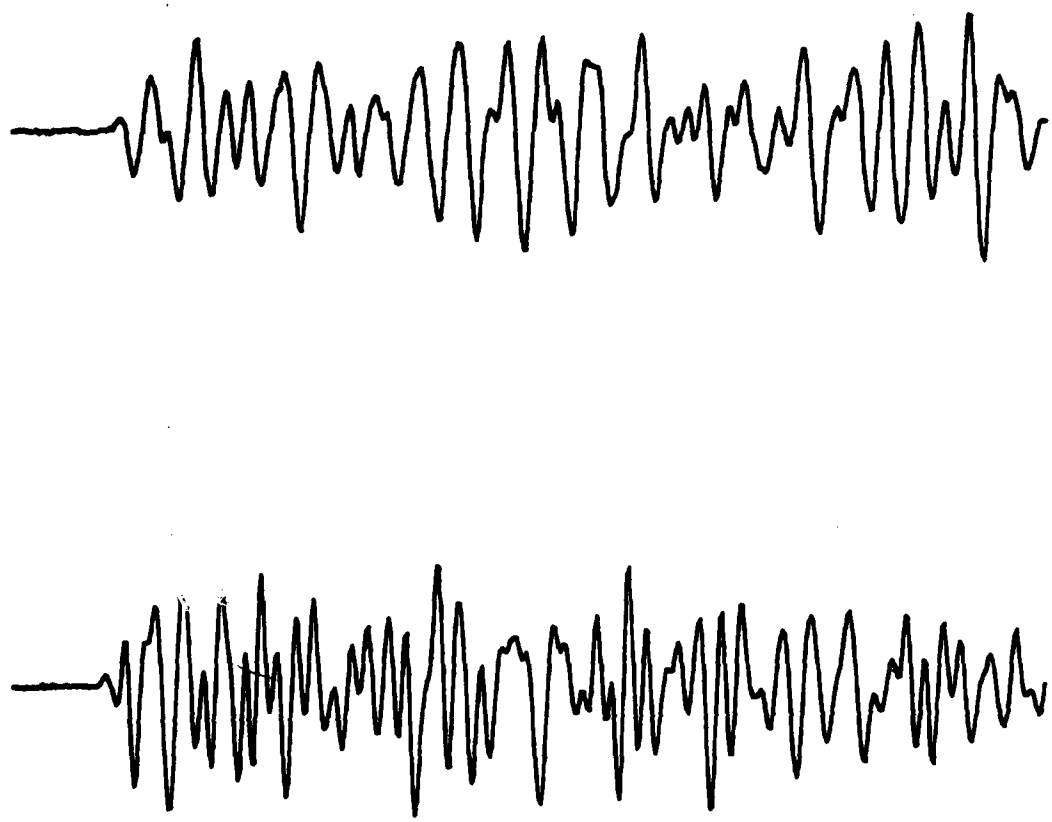


Figure 1.3 Another example of extreme case. The bottom one is a typical earthquake waveform; the top one is an explosion record which looks like an earthquake.

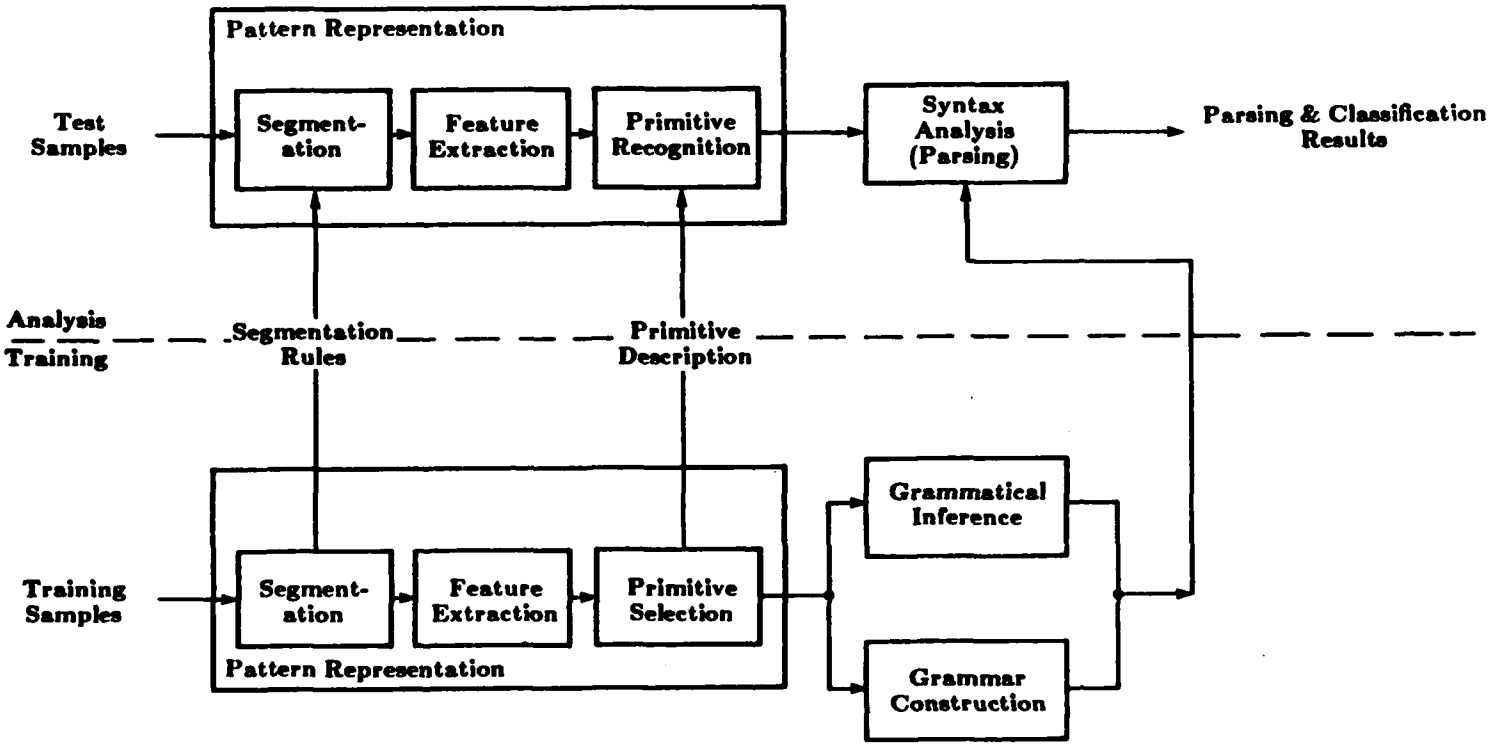



Figure 1.4 Block diagram of a syntactic pattern recognition system.

an exception in many practical applications, and seismic analysis happens to be one of them. This is due to the noise and uncertainty of the source and background. In addition to grammatical approach, we will also use nearest-neighbor decision rule for classification. Of course, the distance, or similarity, computation is between the string representation of the seismic signals. The block diagram of nearest-neighbor classifier for syntactic patterns is shown in Figure 1.5.

Due to the recent advance of VLSI technology it is now feasible and will soon become economical to design custom chips for special applications (Mead and Conway, 1980; Kung, 1979; Ackland, et al., 1981). A VLSI system for seismic signal recognition will also be developed in this study.

## 1.2 Literature Survey

### 1.2.1 Syntactic Pattern Recognition and Digital Signal Processing

Applications of syntactic pattern recognition to digital signal processing have received much attention and achieved considerable success in the past decade (see Fu, 1982). The most prominent applications are in the areas of biomedical waveform analysis and speech recognition. The reason of their success is that these waveforms have regular and predictable structure. Most biomedical waveforms, e.g., ECG wave and carotid pulse wave, are rhythmic and generated by specific organs of the body where their functions are well understood.

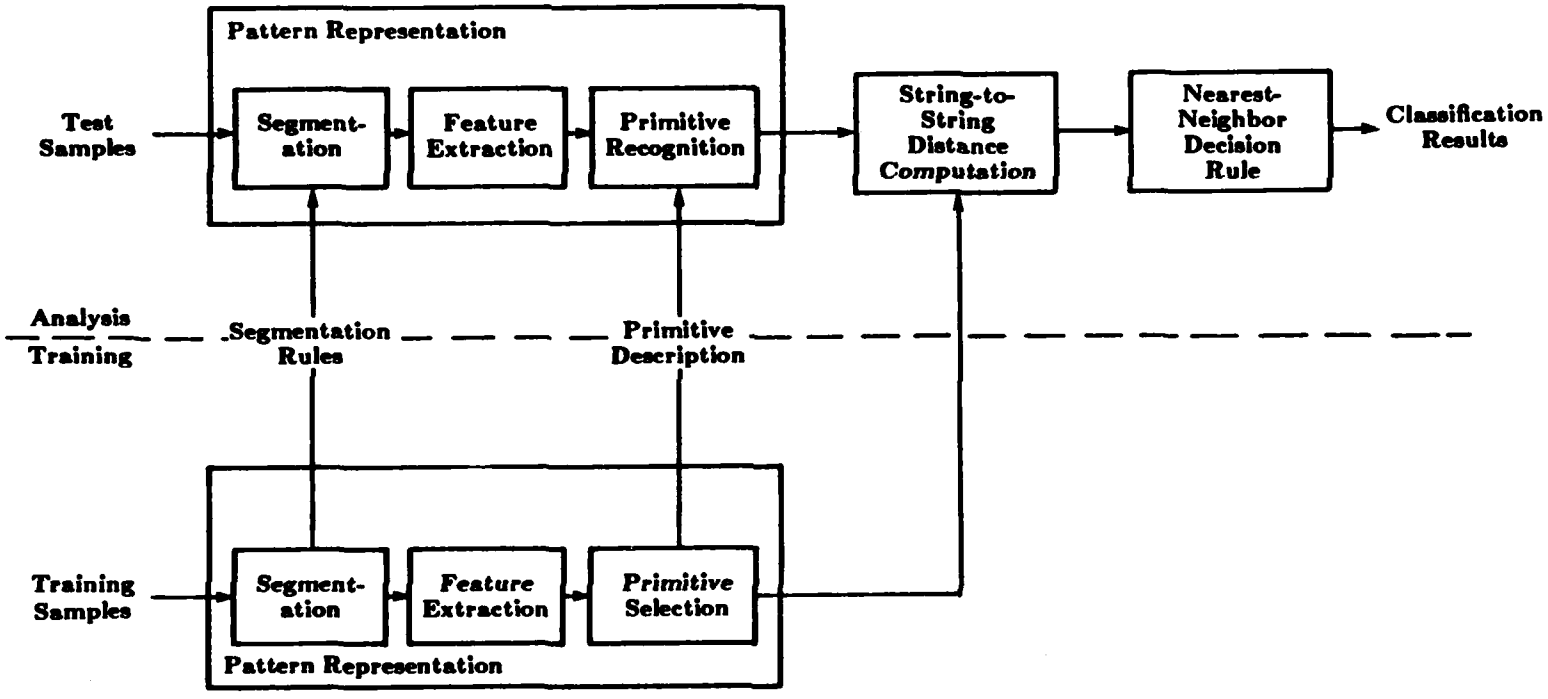



Figure 1.5 Block diagram of a syntactic pattern recognition system using the nearest-neighbor decision rule for string patterns.

It is thus easy to write a grammar for these waveforms based on their functions. Horowitz (1975, 1977) developed a syntactic algorithm to detect the peaks of ECG waves. Albus (1977) used a stochastic finite-state model to interpret ECG signals. Giese, et al., (1979) proposed a syntactic method to analyze EEG signals. Stockman, et al., (1976) applied a syntactic method to analyze carotid pulse waveforms. The major problem in biomedical waveform analysis is the noise which could be generated by muscles or other sources (Albus, 1977).

It has been shown that speech patterns are related to linguistic items by a complex set of rules belonging to "grammar of speech" (DeMori, 1977). Therefore, the most effective way of detecting and recognizing speech patterns is by syntactic method. DeMori (1972) has shown a syntactic method to recognize spoken Italian digits. The major problem in speech recognition is the variability of the speech patterns. They are speaker-dependent as well as context-dependent. Even for the same speaker and the same word, the features extracted from different utterances are usually not the same.

We will review in this section some of the existing syntactic methods applied to signal processing. Although preprocessing is also important, we do not include this part here, because it is case dependent and is usually not related to the recognition stage. However, we will discuss the preprocessing procedure later in our experiments of seismic signal recognition. We will now concentrate on the major parts of syntactic pattern recognition system, i.e., segmentation, feature extraction, primitive selection, grammatical inference or construction, and syntax analysis.

A waveform must be converted into a string of primitives (tree or graph for high dimensional representation) before grammatical inference and syntax analysis can take place. Since a waveform is a one-dimensional signal, it is most natural to represent it by a string of primitives. Various series expansion, for example, Fourier series, and spectral analysis techniques have been used to approximate the whole waveform. However, they are not suitable for syntactic analysis because the relationships among one part of the waveform and the others are significant in syntactic analysis. Although they can be used to feature waveform segment, they are subject to the constraint of segment length and characteristics of the waveform. Pavlidis (1971, 1973, 1974) proposed a linguistic waveform analysis algorithm in which he partitioned the waveform into several segments by using linear approximation. The basic idea is to minimize the number of segments by merging and splitting while the error norm of each segment is retained below the error tolerance. Horowitz (1975, 1977) extended this idea and added peak detection algorithm. He gave a syntactic definition to the positive peak - a positive slope followed by a negative slope or positive slope followed by zero slope and then followed by negative slope. A negative peak can be defined in a similar way. He further constructed a deterministic context-free grammar to recognize positive and negative peaks. This approach is useful in waveform shape analysis because of its simplicity. However, the curvature informations are not included.

Another interesting representation of waveform is by tree structure. It was first introduced by Ehrich and Foith (1976). The peaks and valleys of the waveform are detected and connected by a relational tree. Sankar and Rosenfeld (1979) extended this idea by using the

concepts of fuzzy connectedness. This method converts one-dimensional waveform into two-dimensional tree structure. It is useful for unipolar waveform analysis such as terrain analysis, but not so helpful for the analysis of bipolar waveforms such as ECG wave and random waveforms such as EEG and seismic waves. Another well-known method of converting one-dimensional signal into two-dimensional image is called spectrogram which is used very often in speech analysis (Flanagan, 1972). The spectrogram of a waveform is the plot of energy as a function of time and frequency. Time and frequency are the horizontal and vertical axes of the picture. Energy is represented by gray level intensity. This method needs special facilities to convert a small segment of time-domain signal into frequency-domain representation efficiently. Automatic interpretation of the two-dimensional image is still a subject for studies.

Giese et al. (1979) proposed a syntactic method to analyze EEG signal. The EEG recording is divided into fixed-length segments, each segment is equal to 1-second period. Seventeen features are computed from the spectral of each segment. A linear classifier is applied to classify the segments into seven categories. An EEG grammar is manually constructed and a bottom-up parser without backtracking is used for syntax analysis.

Stockman et al. (1976) proposed a syntactic pattern recognition system for carotid pulse wave analysis. A set of thirteen primitives including various type of line segments and parabolas are used. The subpattern and primitive extraction starts from the most prominent substructure, e.g., long line segment, and then less prominent structures with respect to the more prominent ones, in a prespecified order.

A context-free grammar is manually constructed and a top-down parser is used for syntax analysis.

De Mori (1972, 1977) proposed a syntactic method to recognize spoken digits. First, each 20-msec segment was sent to a low pass filter and a high pass filter, and zero-crossing intervals obtained at the output of the two filters were classified into certain groups, i.e., eight for LPF and five for HPF. Then, each spoken word is represented pictorically on a two-dimensional plane. Finally, a context-free grammar is constructed and a bottom-up parsing is applied. He further introduced syntactic methods for preprocessing, feature extraction, emission and verification of hypothesis and automatic learning of spectral features.

Mottl' and Muchnik (1979) declared that there are two kinds of curve sources which require the linguistic approach for analysis. One kind of source is consistent with the phenomenon which is a process of many stages. The curve consists of parts corresponding to the stages. The junction of the parts are the time when stages change. The segmentation algorithm should divide the curve into a number of adjacent parts characterized by the curve shape. Examples of this kind are ECG waveform and carotid pulse waveform analysis.

The other kind of source represents an object which is chiefly in an invariable state and occasionally leaves as a result of short-time disturbances. For such a curve the segmentation should identify only certain fragments which are regarded as informative while the remainder are left out. Example of this kind is the acoustical diagnosis of internal-combustion engines (Mottl' and Muchnik, 1979).

We feel that seismic wave is the third kind of curve which does not fall exactly into any of the above two categories. The seismic waves are

influenced largely by background as well as by source. Sometimes we are interested in the background, e.g., oil exploration; sometimes we are interested in the source, e.g., nuclear test detection. This will be discussed in the next section.

### 1.2.2 Pattern Recognition and Seismic Signal Analysis

The major studies of seismic waves can be classified into the following areas (Bath, 1979):

*1. Seismic prospecting*. This is the most attractive topic in these days. Seismic methods are applied to exploration for occurrences of oil, ore bodies, minerals, etc. The reflection method and the refraction method are two major methods in use. It should be noted that it is not possible, at least by now, to detect oil, etc., by seismic or any other geophysical methods. It is only possible to discover geological formation which may indicate the occurrence of oil, etc.

*2. Vibration measurements*. The effect of vibraions, due to mining, traffic, etc., on various structures and human beings is studied. Such measurements are usually made with accelerographs.

*3. Stress measurements*. Measurements of absolute stress have been used to investigate the strength of building materials and stability in mines.

*4. Earthquake engineering*. This field studies the effects of earthquakes on all kinds of building structures, especially on crucial structure such as nuclear power plant.

*5. Earthquake prediction.* A very importat field although no significant progress has been made.

6. From the recording of seismic waves to establish the nature of the source. For example:

a) Nuclear test detection - detect secret underground nuclear explosion.

b) Seismic detection of rockburst - locate small rupture by seismic methods.

Most of the existing pattern reconition applications in seismic analyses are to the classification of earthquake and nuclear explosion. Chen (1978) proposed a statistical pattern recognition method for classification of earthquake and nuclear explosion by the seismic wave recording. He emphasized on the extracton of effective features. Geophysical features such as complexity, spectral ratio and third moment of frequency are tested first. Then he used complex cepstrum, orthogonal transformation, autocovariance features and short-time spectral features for classification. His conclusion is that the performance from a single class of features is somehow limited and the combination of various features does not improve the performance because of correlation. He suggested to use both statistical and structural features.

Tjostheim (1975, 1977, 1978) suggested that autoregressive coefficients can be used as features. He has shown that a seismic P-wave can be represented by an autoregressive model of finite order. The short-period P-wave is divided into five segments. The first three autoregressive coefficients of each segment form the feature vector. The combination of different segments is used to achieve better performance. This approach where the whole P-wave is divided into several

segments is an improvement, but still no structural information has been used.

Sarna and Stark (1980) also used autoregressive modeling for feature extraction, but k-nearest neighbor rule for classification. When applied to artificial data, this procedure gave excellent results; however, the results on real seismic / explosion data are very poor. This may indicate that autoregressive modeling is not suitable for real seismic waves. Most of these studies concentrated on feature selection. Only simple decision-theoretic techniques have been used. However, syntactic pattern recognition appears to be quite promising in this area. It uses the structural information of the seismic wave which is very important in analysis.

Syntactic pattern recognition has been pointed out as a promising approach to seismic classification (Chen, 1978). While quite a few statistical approaches have been proposed, we are the first to apply syntactic approaches to this area. With only very simple features, our approaches attain encouraging results comparable to those of the best statistical approaches. Our approaches also differ from the foregoing syntactic methods in the treatment of primitive selection and grammar construction. A clustering procedure along with some decision criteria constitute the primitive selection algorithm in our approach, while heuristic approaches were used by others. Our pattern grammars are inferred from the training samples, but most pattern grammars for signal analysis are constructed manually. An attributed grammar for our specific application is proposed, which could significantly reduce the grammar size and increase the flexibility of description. Finally,

VLSI architectures are proposed for seismic classification, which include feature extraction, primitive recognition and string matching. Our string matcher is different from many contemparory implementations, i.e., exact matching, which are not suitable for pattern recognition applications; the detail will be discussed in chapter V. The results can be produced at a constant rate, i.e., constant time complexity, when using our VLSI architectures with pipelined data flow. Although these VLSI systems are developed for seismic classification, they can be applied to other similar applications.

After the crunch of energy crisis, searching for oil has become more desperate than ever before. Seismological methods use small chemical explosions to generate seismic waves. These seismic waves penetrate down the crust and are reflected by the boundary of different layers. Analysis based on the reflected seismic waves can find the clue about the local crust structure and oil deposit (Bath, 1979). Bois (1981) has applied pattern recognition technique to petroleum prospection.

Another potential field for application of pattern recognition is the damage assesment in structural (earthquake) engineering (Fu and Yao, 1979; Yao, 1979). During a strong earthquake, the accelerometers in a large building structure will record the acceleration of the building. From these recordings (and other informations) the damage of the building, as far as the structural damage reflected on the seismic recordings is concerned, can be classified into certain classes.

## 1.3 Organization of Thesis

We have seen some examples which apply syntactic approach to digital signal analysis. Those systems are usually heuristically constructed and therefore application dependent. We also showed several statistical pattern recognition approaches to seismic classification. We would like to study in this research the application of syntactic pattern recognition to seismic classification. Two approaches are investigated: one uses grammatical inference and error-correcting parsing; the other computes string-to-string distance and applies nearest-neighbor decision rule. Chapter II discusses the string similarity measures, which are hierarchically classified, and classification procedures which include error-correcting parsing and nearest-neighbor decision rule. Chapter III shows procedures and experimental results of syntactic pattern recognition applications to seismic discrimination, i.e., earthquake / explosion classification, and damage assesment. Attributed grammar which can reduce the complexity of the pattern grammar is discussed in Chapter IV. Chapter V discusses VLSI architectures for syntactic seismic classification, which include feature extraction, primitive recognition and string matching. Chapter VI is the summary, conclusion and recommendations for future research.

Although our experiments are on seismic discrimination, these approaches can be applied to other similar applications. For example, pattern recognition method has been used to determine the nature of reserviors in petroleum prospection (Bois, 1981). The unknown reservior is compared with a known reservior, for example, one which contains oil. Features are computed from the seismic traces of the two reserviors and ploted on a two-dimensional plane. The similarity

between the two reserviors is determined by the distribution of the two clusters. Since the nature of the reservior is characterized by the seismic traces, it is possible to compare the seismic traces of the two reserviors directly.

Levenshtein distance has recently been applied to speech recognition (Okuda, Tanaka and Kasai, 1976; Ackroyd, 1980). It can be used to correct the letter or phoneme sequences that are generated by the recognition machine, or can be built directly into the recognition procedures. Our VLSI string matcher can be applied to both cases. Futhermore, our primitive recognizer can also be applied to the case in Ackroyd (1980). Mottl' and Muchnik (1979) proposed a linguistic approach to the analysis of experimental curves where a special-purpose language is constructed to describe the pattern. The distance between two strings is defined as the minimum number of insertion and deletion of symbols, which is in essence equivalent to Levenshtein distance.

# CHAPTER II

# SIMILARITY MEASURES AND RECOGNITION PROCEDURES FOR STRING PATTERNS

## 2.1 Introduction

One important premise in pattern recognition is that we can measure the similarities between patterns. We say that a pattern belongs to one class if and only if that pattern is more similar to the members of this class than the members of other classes. These measures can be nominal where numbers used only as names, or ordinal where only rank orders have meaning, or interval where seperation between numbers is meaningful, or ratios where a natural zero exists. Distance is a popular candidate for simlarity measure. If the pattern is represented by a vector, as in the case of statistical approach, the Euclidean distance is usually used as a similarity measure. The Euclidean distance has many nice properties, for example, symmetric and invariant under translation and rotation.

In syntactic approach, patterns are represented by strings, trees or graphs, therefore similarity measures must be available for these syntactic patterns. Several similarity measures have been proposed to tackle this problem (Fu, 1977; Lu and Fu, 1977, 1978b). Since our major interest is string patterns, we will review some well-known string similarity measures, discuss their properties and define a hierarchy of

string distances.

String similarity measure can be applied to string-matching in information storage and retrieval (Hall and Dowling, 1980), speech recognition (Sakoe and Chiba, 1978), clustering of string patterns (Fu and Lu, 1977) and nearest-neighbor decision rule for string classification. It is also used in error-correcting parsing. Given a string $y$ and a language $L(G)$, an error-correcting parser (ECP) generates a parse for string $x$, where $x \in L(G)$ and $x$ is most similar to $y$.

Section 2 of this chapter discusses various types of string similarity measures, including both nonstochastic and stochastic models. String distances are classified into general string distances and special string distances. General string distances are based on the principles of insertion, deletion and substitution transformations. Special string distances are those not based on the above principles. One example is the time warping distance in speech analysis. We propose another special distance computation for damage assesment. A hierarchy of general string distances are also defined. Section 3 describes error-correcting parsing algorithms which do not require expanded grammars. Section 4 discusses and compares two recognition procedures, namely, the error-correcting parsing and the nearest-neighbor rule, for syntactic patterns, and Section 5 gives the conclusion.

This chapter emphasizes the symmetric property of string similarity measures. This is not a problem when we use Euclidean distance as the similarity measure, since Euclidean distance is always symmetric. But this is not true when we define string similarity measures, especially when using weighted distance. The error-correcting parsing algorithms using symmetric string similarity measures are also given which

can not be solved by any other existing parsing algorithm.

## 2.2 Similarity Measures of Strings

String similarity measures can be defined in terms of two different concepts, i.e., distance concept and likelihood concept. The former is for nonstochastic models and the latter is for stochastic models. Consider string $x = a_1 a_2 \cdots a_n$ and string $y = b_1 b_2 \cdots b_m$, the string similarity measure between $x$ and $y$ is defined as the distance or probability that string $y$ is transformed from string $x$. The distance or probability of transformation from $x$ to $y$ is ususlly different from that of transformation from $y$ to $x$, therefore, results in an asymmetric similarity measure, i.e., the similarity between $x$ and $y$ is different from the similarity measure between $y$ and $x$. This is a big disadvantage in some applications, for example, in string clustering. The inconsistency in similarity measures makes the outcome inconsistent. Therefore we want to discuss the symmetric property of the string similarity measure.

### 2.2.1 Similarity Measures based on Distance Concept

The distance measures between strings have been proposed for more than one decade and appeared often in the literature (see Fu, 1982). It is known (Okuda, et al., 1976) that Weighted Levenshtein Distance (WLD) is more accurate in the correction of string errors than the abbreviation method (Blair, 1960), the ordered key letters method (Tanakd and Kasai, 1972) and the elastic matching method

(Levenshtein, 1966), where all of these apply substitution, insertion and deletion to string symbols. Fu and Lu (1977) have classified the weight metrics into three categories, but did not consider the symmetric property of the metric. We would like to further extend this idea and include the discussion of symmetric property.

*A. General String Distances*

One of the primitive string distance definitions is called the Levenshtein distance (Levenshtein, 1966). The Levenshtein distance between strings $x$ and $y$, $x, y \in \Sigma^*$, denoted as $d^L(x,y)$, is defined as the smallest number of transformations required to derive string $y$ from string $x$. The transformations include insertion, deletion and substitution of terminal symbols.

*Definition 2.1* For any two strings $x, y \in \Sigma^*$, we can define a sequence of transformations $J=\{T_1, T_2, \ldots, T_n\}$, $n \geq 0$, $T_i \in \{T_S, T_D, T_I\}$ for $1 \leq i \leq n$, such that $y \in J(x)$. The transformations $T_S$, $T_D$ and $T_I$ are defined as follows:

(1) substitution transformation, $T_S$

$$\omega_1 a \omega_2 \,|\xrightarrow{T_S}\, \omega_1 b \omega_2 \text{ for all } a,b \in \Sigma,\ a \neq b.$$

(2) deletion transformation, $T_D$

$$\omega_1 a \omega_2 \,|\xrightarrow{T_D}\, \omega_1 \omega_2 \text{ for all } a \in \Sigma.$$

(3) insertion transformation, $T_I$

$$\omega_1 \omega_2 \,|\xrightarrow{T_I}\, \omega_1 a \omega_2 \text{ for all } a \in \Sigma.$$

where $\omega_1, \omega_2 \in \Sigma^*$.

*Definition 2.2* The Levenshtein distance $d^L(x,y)$ is defined as

$$d^L(x,y) = \min_j \left\{ k_j + m_j + n_j \right\}$$

where $k_j$, $m_j$ and $n_j$ are respectively the number of substitution, deletion and insertion transformations in $J$.

*Definition 2.3* A distance between two strings $x, y \in \Sigma^*$, $d(x,y)$ is symmetric if and only if $d(x,y) = d(y,x)$.

Since all the insertion, substitution and deletion transformations are counted equally, the Levenshtein distance is symmetric. It is equivalent to assigning weight 1 to each of the transformation. We call these weights type 0 weights.

The computation of the Levenshtein distance can be implemented by dynamic programing technique on a grid matrix as shown in Figure 2.1. The partial distance $\delta[i,j]$, which denotes the minimum distance from point (0, 0) to point $(i,j)$, can be computed from the partial distances $\delta[i,j-1]$ $\delta[i-1,j-1]$ and $\delta[i-1,j]$ as shown in Figure 2.2. The total distance is simply $\delta[n,m]$, where $n$ is the length of the reference string and $m$ is the length of the test string.

Since the minumum distance is unlikely to occur in some areas of the grid matrix, for example, the upper left corner and lower right corner, a globol path constraint can be imposed to save computation time. Figure 2.3 shows a window constraints such that only those points $(i, j)$, $|i - \frac{n}{m}j| \leq r$, where $0 \leq i \leq n$, $0 \leq j \leq m$, $r$ is a selected constant, are subject to distance computation. Algorithm 2.1 is for general string distance computation with global path constraint.

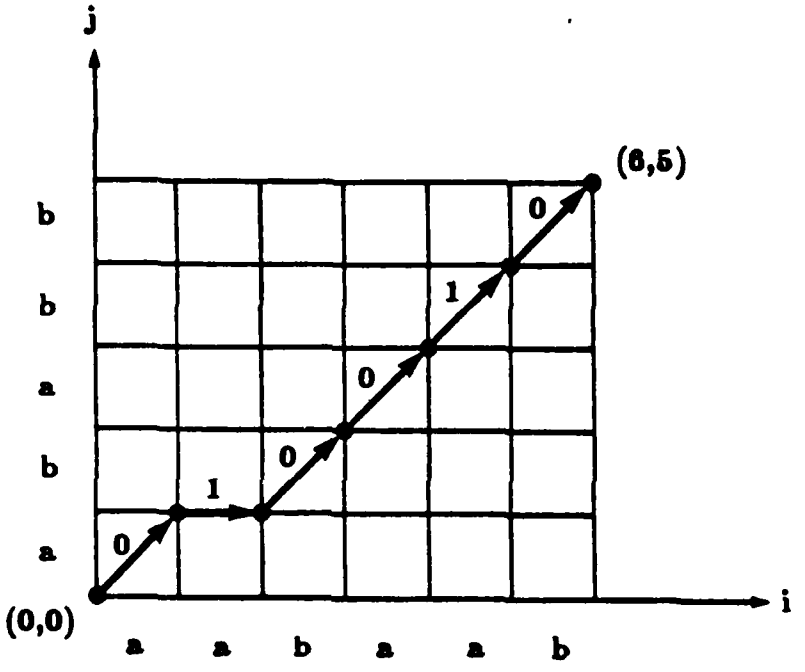



Figure 2.1 The transformation from string 'aabaab' to 'ababb'. The Levenshtein distance $d^L(aabaab, ababb) = 2$.

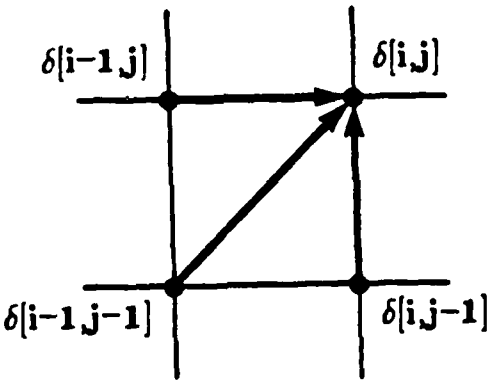



Figure 2.2 The partial distance $\delta[i,j]$ is computed from $\delta[i,j-1]$, $\delta[i-1,j-1]$ and $\delta[i-1,j]$.

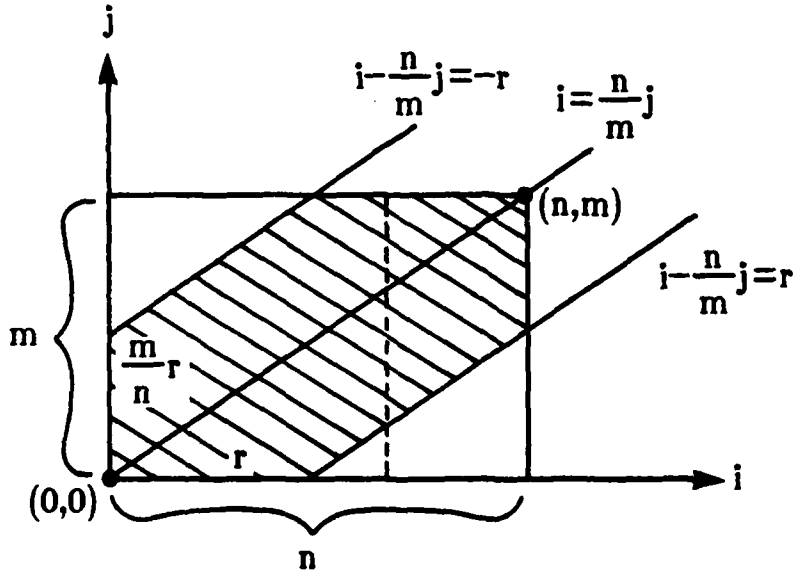



Figure 2.3 An example of global path constraint.

*Algorithm 2.1.* Computation of general string distance with global path constraint

*Input:* Two strings $x=a_1a_2\cdots a_n$ and $y=b_1b_2\cdots b_m$ where $a_i, b_j \in \Sigma$ for all $1\le i\le n$, $1\le j\le m$, and a constant $r$ for global path constraint.

*Output:* The general string distance $d(x,y)$.

*Method:*

(1) $\delta[0, 0] := 0$;

(2) for $i := 1$ to $r$ do $\delta[i, 0] := \delta[i\text{-}1, 0] + \Delta_i$;

(3) for $j := 1$ to $\frac{m}{n}r$ do $\delta[0,j] := \delta[0,j-1] + \Delta_j$;

(4) for $j := 1$ to $m$ do begin

$i1 := \frac{n}{m}j$ - r;

$i2 := \frac{n}{m}j$ + r;

for $i := i1$ to $i2$ do

if $(i \ge 1)$ and $(i\le n)$ then $\delta[i,j] := min(i,j)$;

(* $min(i,j)$ is a function for local distance computation *)

end;

(5) $d(x,y) := \delta[n,m]$;

We use a function $min(i,j)$ in Algorithm 2.1 to compute the local distance. The function $min(i,j)$ can be computed seperately to match different local distance constraints and return a distance value. For Levenshtein distance, $min(i,j)$ = min { $\delta[i-1,j] + 1$, $\delta[i,j-1] + 1$, $\delta[i-1,j-1] + 1$ } if $a_i \neq b_j$; otherwise $min(i,j) = \delta[i-1,j-1]$. This arrangement is more flexible since the dynamic programming portion never need change, only different function $min(i,j)$ is used for

different applications.

The Levenshtein distance appears to be not powerful enough for many pattern recognition applications. However, it may be sufficient for string matching in information retrieval (Hall and Dowling, 1980). Fu and Lu (1977) have proposed a weighted Levenshtein distance (WLD) where different weights are associated with different transformation and terminals.

We can make the string distance definition more flexible and practical by assigning different weights to different transformations and/or terminals. There are at least three possible cases. In the first case, different weights are assigned to different transformations but all terminals are treated equally. We call these weights type 1 weights. Here are the transformations:

(1) $\omega_1 a \omega_2 \mid \xrightarrow{T_S,\ \sigma} \omega_1 b \omega_2$ for all $a, b \in \Sigma$, $a \neq b$, where $\sigma$ is the cost of substituting $b$ for $a$, $\sigma = 0$ when $b = a$.

(2) $\omega_1 a \omega_2 \mid \xrightarrow{T_D,\ \gamma} \omega_1 \omega_2$ for all $a \in \Sigma$, where $\gamma$ is the cost of deleting $a$.

(3) $\omega_1 \omega_2 \mid \xrightarrow{T_I,\ \rho} \omega_1 a \omega_2$ for all $a \in \Sigma$, where $\rho$ is the cost of inserting $a$.

where $\omega_1, \omega_2 \in \Sigma^*$.

The distance defined by these transformations is called *type* 1 weighted Levenshtein distance.

*Definition 2.4:* The type 1 weighted Levenshtein distance $d^{W1}(x,y)$ is defined as

$$d^{W1}(x,y) = \min_j \left\{ \sigma \cdot k_j + \gamma \cdot m_j + \rho \cdot n_j \right\}$$

where $k_j$, $m_j$ and $n_j$ are defined the same as in Definition 2.3.

*Theorem 2.1* $d^{W1}(x,y)$ is symmetric, i.e., $d^{W1}(x,y) = d^{W1}(y,x)$, if and only if $\gamma = \rho$.

The WLD $d^{W1}(x,y)$ can be computed by Algorithm 2.1 where $min(i,j) = \min \{\delta[i,j-1] + \rho, \delta[i-1,j-1] + \sigma, \delta[i-1,j] + \gamma\}$ as shown in Figure 2.4(a). The weights in step (2) and (3) should also be changed.

In the second case, different weights are assigned to different transformations and terminals, but the weights associated with the terminals are context-indepentent. We call these weights type 2 weights. We have the following transformations:

(1) $\omega_1 a \omega_2 \mid \frac{T_S,\ S(a,b)}{} \omega_1 b \omega_2$ for all $a,b \in \Sigma$, $a \neq b$, where $S(a,b)$ is the cost of substituting $b$ for $a$, $S(a,a) = 0$.

(2) $\omega_1 a \omega_2 \mid \frac{T_D,\ D(a)}{} \omega_1 \omega_2$ for all $a \in \Sigma$, where $D(a)$ is the cost of deleting $a$.

(3) $\omega_1 \omega_2 \mid \frac{T_I,\ I(a)}{} \omega_1 a \omega_2$ for all $a \in \Sigma$, where $I(a)$ is the cost of inserting $a$.

where $\omega_1, \omega_2 \in \Sigma^*$.

The distance defined by these transformations is called *type* 2 weighted Levenshtein distance.

*Definition 2.5.* The type 2 weighted Levenshtein distance $d^{W2}(x,y)$ is defined as

$$d^{W2}(x,y) = \min_j \left\{ \sum S_j(a,b) + \sum D_j(a) + \sum I_j(a) \right\}$$

where $a,b \in \Sigma$ and $J$ is the sequence of transformations used to derive $y$ from $x$.

*Theorem 2.2* $d^{W2}(x,y)$ is symmetric if and only if $D(a) = I(a)$ and $S(a,b) = S(b,a)$ for all $a,b \in \Sigma$.

The type 2 WLD $d^{W2}(x,y)$ can also be computed by Algoritm 2.1 where $min(i,j) = \min \{\delta[i,j-1] + I(b_j),\ \delta[i-1,j-1] + S(a_i,b_j),\ \delta[i-1,j] + D(a_i)\}$ as shown in Figure 2.4(b)..

In the third case, the weights associated with the terminals for insertion and deletion are context-dependent. We call these weights type 3 weights. We have the following transformations:

(1) $\omega_1 a \omega_2 \mid \xrightarrow{T_S,\ S(a,b)} \omega_1 b \omega_2$ for all $a,b \in \Sigma$, $a \neq b$, where $S(a,b)$ is the cost of substituting $b$ for $a$, $S(a,a) = 0$.

(2) $\omega_1 ab \omega_2 \mid \xrightarrow{T_D,\ D(b,a)} \omega_1 b \omega_2$ for all $a \in \Sigma$, $b \in \Sigma \cup \{\&\}$, where $D(b,a)$ is the cost of deleting $a$ in front of $b$.

(3) $\omega_1 a \omega_2 \mid \xrightarrow{T_I,\ I(a,b)} \omega_1 ba \omega_2$ for all $b \in \Sigma$, $a \in \Sigma \cup \{\&\}$, where $I(a,b)$ is the cost of inserting $b$ in front of $a$.

where $\omega_1, \omega_2 \in \Sigma^*$.

The reason of using (2) is for symmetric purpose. As we mentioned earlier, the symmetric property is important in distance computation; otherwise, the distance between two strings will not be unique, depending on the selection of reference string and test string. In string recognition, there may not be such problem, since we know the reference

and test string. However, in string clustering, the problem will occur, since we have to treat each string equally. Context-dependent weights are useful in some other applications, for example, in speech recognition, where the repetition of some symbols is considered legal. For instance, the strings $x$, $y$, where

$x = a\ a\ a\ b\ b\ c$

$y = a\ a\ b\ b\ c\ c$

may be considered identical, i.e., with zero distance. In this case, it can be easily implemented by letting $I(a,a) = D(a,a) = 0$ for all $a \in \Sigma$.

The distance defined by these transformations is called type 3 weighted Levenshtein distance. These transformations are similar to what Fu and Lu (1977) have proposed but different in two aspects. First, a right endmarker "&" is used for both the reference and test strings, therefore no additional transformations are needed to handle the end point insertion or deletion. From now on, we will use $\Sigma'$ to represent $\Sigma \cup \{\&\}$. Second, the weights associated with deletion transformation are context-depentent.

*Definition 2.6:* The type 3 weighted Levenshtein distance $d^{W3}(x,y)$ is defined as

$$d^{W3}(x,y) = \min_j \left\{ \sum S_j(a,b) + \sum D_j(c,a) + \sum I_j(c,a) \right\}$$

where $a,b \in \Sigma$, $c \in \Sigma'$ and $J$ is the sequence of transformations used to derive $y$ from $x$.

*Theorem 2.3* $d^{W3}(x,y)$ is symmetric if and only if $D(a,b) = I(a,b)$ and $S(a,b) = S(b,a)$ for all $b \in \Sigma$, $a \in \Sigma'$.

Before deriving algorithm for computing type 3 WLD, we have to consider one more problem. Since the weights are context-dependent, the order of insertion and deletion transformations can no longer be ignored.

*Example 2.1:* Let the string $y=\alpha bcda\beta$ and $x=\alpha a\beta$, $x,y \in \Sigma^*$, $\alpha, \beta \in (\Sigma \cup N)^*$, then the transformations from $x$ to $y$ can be

$$\alpha a\beta \mid \xrightarrow{I(a,b)} \alpha ba\beta \mid \xrightarrow{I(a,c)} \alpha bca\beta \mid \xrightarrow{I(a,d)} \alpha bcda\beta, \text{ or}$$

$$\alpha a\beta \mid \xrightarrow{I(a,b)} \alpha ba\beta \mid \xrightarrow{I(a,d)} \alpha bda\beta \mid \xrightarrow{I(d,c)} \alpha bcda\beta, \text{ or}$$

$$\alpha a\beta \mid \xrightarrow{I(a,c)} \alpha ca\beta \mid \xrightarrow{I(c,b)} \alpha bca\beta \mid \xrightarrow{I(a,d)} \alpha bcda\beta, \text{ or}$$

$$\alpha a\beta \mid \xrightarrow{I(a,c)} \alpha ca\beta \mid \xrightarrow{I(a,d)} \alpha cda\beta \mid \xrightarrow{I(c,b)} \alpha bcda\beta, \text{ or}$$

$$\alpha a\beta \mid \xrightarrow{I(a,d)} \alpha da\beta \mid \xrightarrow{I(d,b)} \alpha bda\beta \mid \xrightarrow{I(d,c)} \alpha bcda\beta, \text{ or}$$

$$\alpha a\beta \mid \xrightarrow{I(a,d)} \alpha da\beta \mid \xrightarrow{I(d,c)} \alpha cda\beta \mid \xrightarrow{I(c,b)} \alpha bcda\beta$$

There are six different transformations available for Example 2.1. In fact, there are $k!$ different transformations to insert $k$ symbols in front of any specific symbol such that all have the same final result. In Example 2.1 there is no reason to assume that the order of insertion is "$b$ followed by $c$ followed by $d$". Therefore, the minimum cost transformation should be determined from those six transformations. However, the computation is much more complicated so that the little gain from the real minimum cost transformation may not pay off the extra amount of computation. If we are allowed to chose a suboptimal solution, we will stick to one type of the transformation, i.e., the first one in Example 2.1.

The cases for deletion are similar to those for insertion. Consider Example 2.1, the transformation from $y$ to $x$ corresponding to the first one is as follows:

$$\alpha bcda\beta \mid \underline{D(a,d)}\ \alpha bca\beta \mid \underline{D(a,c)}\ \alpha ba\beta \mid \underline{D(a,b)}\ \alpha a\beta$$

It is noted that the symmetric property is preserved here.

We can use Algorithm 2.1 to compute the type 3 WLD $d^{W3}(x,y)$ where $min(i,j)$ = min { $\delta[i,j-1] + I(a_{i+1},b_j)$, $\delta[i-1,j-1] + S(a_i,b_j)$, $\delta[i-1,j] + D(b_{j+1},a_i)$ } as shown in Figure 2.4(c). The weights in step (2) and (3) should also be modified.

We can define a hierarchy on the four types of distances, i.e., type 0 distance is a proper subset of type 1 distance; type 1 distance is a proper subset of type 2 distance, and type 2 distance is a proper subset of type 3 distance. They are capable of computing all the general string distances based on the concepts of insertion, deletion and substitution transformations. However, there are some exceptions of distance measurements which do not base on the idea of insertion, deletion and subtitution transformations. We will call them the special string distances.

*B. Special String Distance*

The special string distances mean that these distances can only be applied to some specific applications, also they are not based on the idea of insertion, deletion and substitution transformations. One example is the dynamic time warping for speech recognition, the other is the modified dynamic time warping for damage assesment.

In spoken word recognition, the recorded speech signal from different utterance is different even for the same word by the same

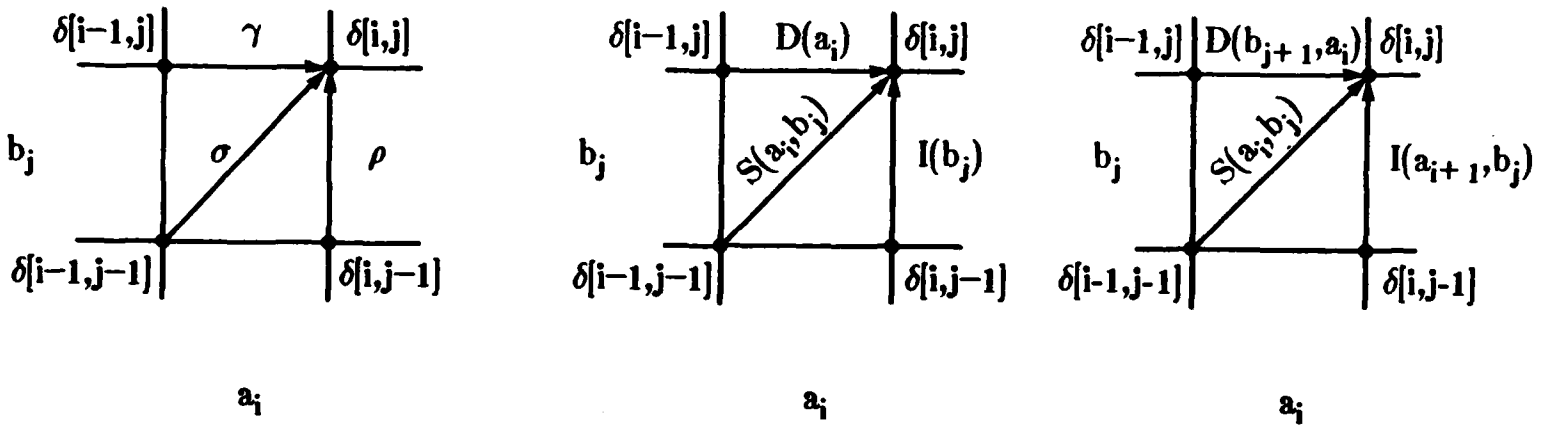



Figure 2.4 Computation of partial distance for (a) type 1, (b) type 2 and (c) type 3 WLD.

person. Meanwhile, the time difference between speech patterns is nonlinear, therefore a nonlinear matching algorithm is requiered in order to obtain good recognition results. A special technique called time warping has been proposed by Sakoe and Chiba (1978). An example is shown in Figure 2.5 where $x = a_1 a_2 \ldots a_n$ is the reference pattern and $y = b_1 b_2 \ldots b_m$ is the test pattern. Each component $a_i$, $b_j$ of string $x$, $y$ can be a feature vector or a scalar which represents a signal segment. (The position of each component $a_i$, $b_j$ in the grid matrix is slightly different from what we have used previously.)

*Definition* 2.7 The time warping distance between strings $x$ and $y$ is

$$d^{TW}(x, y) = \sum_{k=1}^{K} d(c(k))$$

where

$$d(c(k)) = d(i(k), j(k)) = \|a_{i(k)} - b_{j(k)}\|$$

and $k$ is the index of common time axis.

Two major differences between time warping and the general string-to-string distance can be pointed out immediately. First, one component, i.e., symbol, in warping function can be used more than once. For example, component $a_4$ in Figure 2.6 has been used to compared with $b_3$ and $b_4$. Second, the components may be skipped without any cost. Although the general string distance can be modified by letting $I(a,a)=0$ and $D(a,b)=0$ for $a,b \in \Sigma$, to simulate time warping, there are other restrictions on the time warping function, for example, slope constraint. Slope constraint will eliminate excessively steep or gentle gradient from the warping function. For details of slope constraints and computation of time warping distance, see (Sakoe and Chiba, 1978). The weights used for time warping are different from

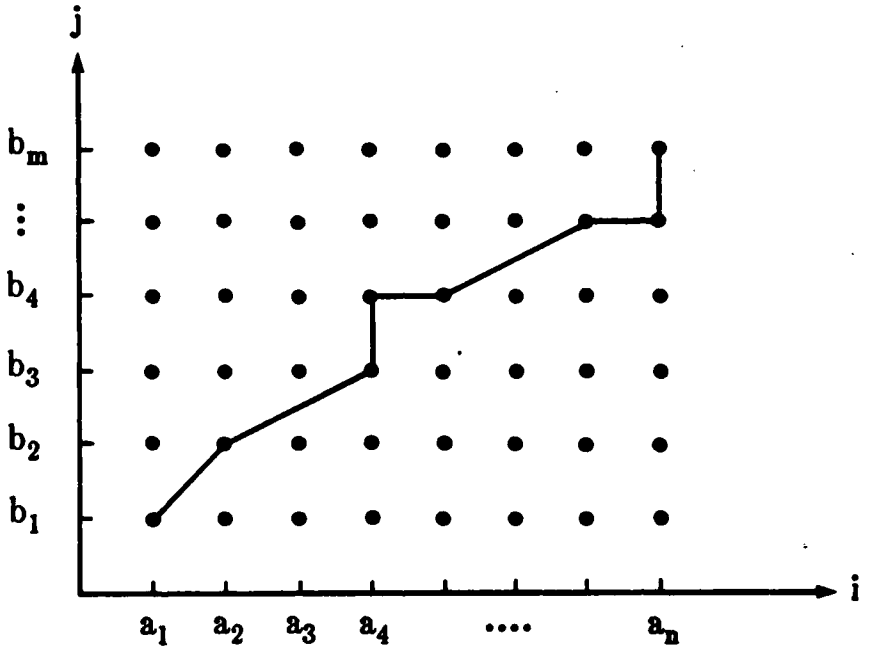



Figure 2.5 An example of dynamic time warping.

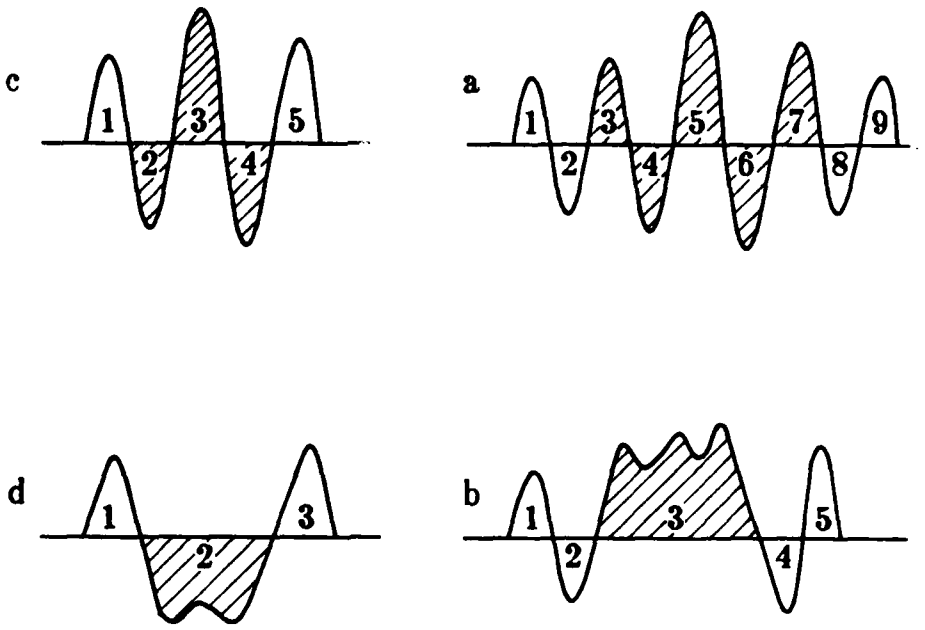



Figure 2.6 Examples of some seismic recordings in structural damage assesment.

those for insertion, deletion and substitution, and can be tailored to fit specific applications.

A path constraint similar to that of general string distance (see Fig 2.3) can also be applied here, i.e.,

$$|i(k)-\frac{n}{m}j(k)| \leq r$$

where $r$ is the path width. This will prevent warping function from having unrealistic matches. Sakoe and Chiba (1978) proposed a path constraint

$$|i(k)-j(k)| \leq r$$

This window is along the diagonal axis $i(k)=j(k)$. Since the dynamic programming proceeds from point (0,0) to point $(n,m)$, the window should be along the diagonal axis $i(k) = \frac{n}{m}j(k)$ as shown in Figure 2.3. It has been shown by Sakoe and Chiba (1978) that the symmetric time warping distance has higher recognition accuracy than asymmetric time warping distance.

In some applications, specifically string distance computation for damage assesment, one component in one string is equivalent to the summation of several components in another string. For example, in Figure 2.6 the top two segments may come from the seismic recordings of a buildings without damage while the bottom two segments may come from the same building with certain degree of damage. If we consider each component in Figure 2.6 as an appropriate measurement then $b_3 = a_3 + a_4 + a_5 + a_6 + a_7$ and $d_2 = c_2 + c_3 + c_4$, since $b_3$ is a distortion of $a_3$, $a_4$, $a_5$, $a_6$ and $a_7$, and $d_2$ is a distortion of $c_2$, $c_3$ and $c_4$.

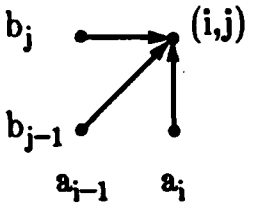
Therefore we can modify the slope constraints and local distance functions in Sakoe and Chiba (1978) and use them for distance computation. The modified slope constraints are shown in Figure 2.7. Since the local distance functions $min(i,j)$ are symmetric, the modified time warping distance is also symmetric. The local distance functions $min(i,j)$ are changable as we will see in chapter III.

*C. Normalized Distance*

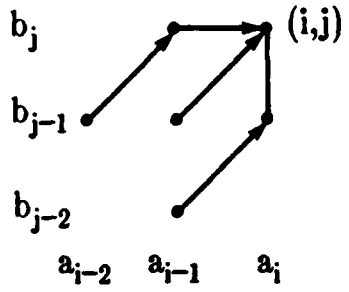
All the distance measures discussed so far are absolute distances. For example, consider two pairs of strings $x_1$, $y_1$ and $x_2$ and $y_2$,

$$x_1 = aaabbbcccddd$$
$$y_1 = acabbbcccdbd$$
$$x_2 = ad$$
$$y_2 = cb$$

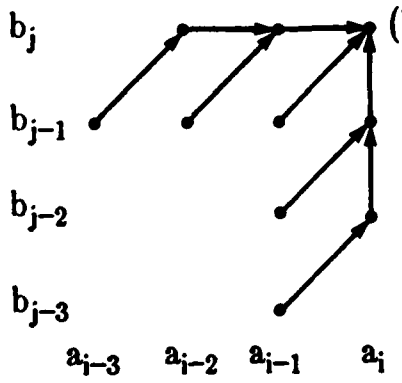
The distance between $x_1$ and $y_1$ is two (substitution errors). The distance between $x_2$ and $y_2$ is also two (substitution errors). However, when taking the whole string length into consideration, string pair $x_1$ and $y_1$ are more similar than string pair $x_2$ and $y_2$. This shows that equal absolute distance does not necessarily indicate equal similarity. Sakoe and Chiba (1978) have proposed a normalized distance for dynamic time warping, which is equal to division of the absolute distance by the total length of the strings. When absolute distances are equal, the normalized distances tend to favor longer strings. This same idea can be applied to general string distance computation with insertion, deletion and substitution.




$$\delta[i,j] = \min \begin{bmatrix} \delta[i,j-1] + |a_i - b_j| \\ \delta[i-1,j-1] + |a_i - b_j| \\ \delta[i-1,j] + |a_i - b_j| \end{bmatrix}$$




$$\delta[i,j] = \min \begin{bmatrix} \delta[i-1,j-2] + |a_i - b_j - b_{j-1}| \\ \delta[i-1,j-1] + |a_i - b_j| \\ \delta[i-2,j-1] + |a_{i-1} + a_i - b_j| \end{bmatrix}$$




$$\delta[i,j] = \min \begin{bmatrix} \delta[i-1,j-3] + |a_i - b_j - b_{j-1} - b_{j-2}| \\ \delta[i-1,j-2] + |a_i - b_j - b_{j-1}| \\ \delta[i-1,j-1] + |a_i - b_j| \\ \delta[i-2,j-1] + |a_{i-1} + a_i - b_j| \\ \delta[i-3,j-1] + |a_{i-2} + a_{i-1} + a_i - b_j| \end{bmatrix}$$

Figure 2.7 Examples of slope constraints and corresponding local distance function of modified time warping distance.

### 2.2.2 Similarity Measures Based on Likelihood Concept

The string distance measures discussed in the previous section are for nonstochastic models. In stochastic language, every string is associated with a probability (Fu and Huang, 1972). Therefore, we use probability, instead of weight, to characterize the transformation. Some of the stochastic context-dependent transformations have been proposed, for example, substitution has been proposed by Fung and Fu (1975), substitution and insertion have been proposed by Lu and Fu (1977b). Here we add context-dependent deletion transformation. We still use $T_S$, $T_I$ and $T_D$ to represent substitution, insertion and deletion transformation respectively. Associated with $T_S$, $T_I$ and $T_D$ we use $P_S$, $P_I$ and $P_D$ for transformation probabilities. Transformations with context-dependent probabilities are defined as follows:

(1) $\omega_1 a \omega_2 \mid \xrightarrow{T_S,\ P_S(b \mid a)} \omega_1 b \omega_2$ for all $a, b \in \Sigma$, where $P_S(b \mid a)$ is the probability of substituting $b$ for $a$.

(2) $\omega_1 ab \omega_2 \mid \xrightarrow{T_D,\ P_D(b \mid ab)} \omega_1 b \omega_2$ for all $a \in \Sigma$, $b \in \Sigma'$, where $P_D(b \mid ab)$ is the probability of deleting $a$ in front of $b$.

(3) $\omega_1 a \omega_2 \mid \xrightarrow{T_I,\ P_I(ba \mid a)} \omega_1 ba \omega_2$ for all $b \in \Sigma$, $a \in \Sigma'$, where $P_I(ba \mid a)$ is the probability of inserting $b$ in front of $a$.

where $\omega_1, \omega_2 \in \Sigma^*$, and

$$\sum_{b \in \Sigma} P_S(b \mid a) + \sum_{b \in \Sigma'} P_D(b \mid ab) + \sum_{b \in \Sigma} P_I(ba \mid a) = 1$$

for all $a \in \Sigma$.

The probability associated with the transformation of one string from another is called stochastic similarity. A higher transformation probability between two strings means they are more similar. Similar to the various weights for nonstochastic cases in Section 2.2.1, we can also define many different types of transformation probabilities, for example, context independent, terminal independent or transformation independent. Since they are the simplified versions of the one just defined, we will only use the above one as an example in the following.

*Definition* 2.8 The stochastic similarity between strings $x$ and $y$, $d^S(x,y)$, is defined as

$$d^S(x,y) = p(y \mid x)$$

$$= \max_j q_j(y \mid x)$$

where

$q_j(y \mid x)$ is the probability of transfomations $J$ which derives $y$ from $x$.

The transformation probability $p(y \mid x)$ is the maximum probability among those associated with all the possible transformations from $x$ to $y$.

*Theorem 2.4* $d^S(x,y)$ is symmetric if and only if $P_D(a \mid ba) = P_I(ba \mid a)$ and $P_S(b \mid c) = P_S(c \mid b)$ for all $b, c \in \Sigma$, $a \in \Sigma'$.

The computation of stochastic similarity can also be carried out by dynamic programming technique A local probability function replaces the local distance function of nonstochastic cases. However, the probability function selectes the maximum of the probabilities which come from below, left and lower left, see Figure 2.8 for a graphic illustration.

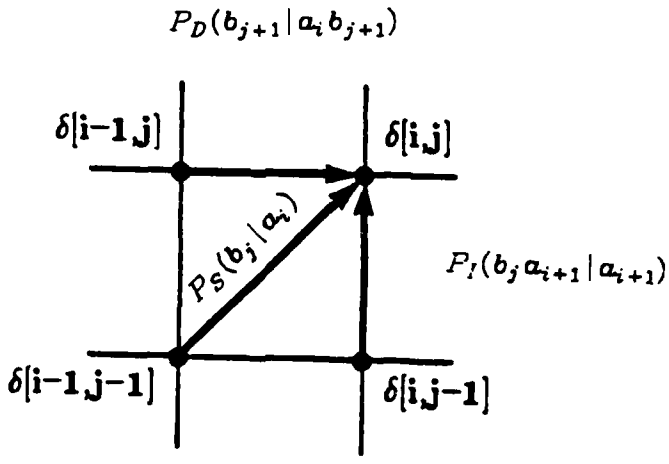



Figure 2.8 Computation of partial distance for stochastic models.

*Algorithm 2.3* Computation of stochastic string similarity

*Input:* Two strings $x = a_1 a_2 \ldots a_n a_{n+1}$ and $y = b_1 b_2 \ldots b_m b_{m+1}$ where $a_i, b_j \in \Sigma$ for all $1 \le i \le n$, $1 \le j \le m$, $a_{n+1} = \&$, $b_{m+1} = \&$, and the probabilities associated with transformations on terminals in $\Sigma$ and $\{\&\}$.

*Output:* stochastic similarity $d^S(x,y)$.

*Method:*

(1) $\delta[0, 0] := 1$;

(2) for $i := 1$ to $n$ do $\delta[i,0] := \delta[i-1,0] \cdot P_D(b_1 | a_i b_1)$;

(3) for $j := 1$ to $m$ do $\delta[0,j] := \delta[0,j-1] \cdot P_I(b_j a_1 | a_1)$;

(4) for $i := 1$ to $n$ do

for $j := 1$ to $m$ do begin

$\delta[i,j] := \max \{\delta[i,j-1] \cdot P_I(b_j a_{i+1} | a_{i+1}),$

$\delta[i-1,j-1] \cdot P_S(b_j | a_i), \delta[i-1,j] \cdot P_D(b_{j+1} | a_i b_{j+1})\}$;

end;

(5) $d^S(x,y) := \delta[n,m]$;

We can also use a global path constraint here to speed up the computation.

Similarity measure is one of the fundamental constituent of pattern recognition. In some applications, for example, string-matching, the recognition accuracy relies almost entirely on the accuracy of similarity measure. Even the error-correcting parsing is closely related to similarity measures. We will discuss the relation between EC (error-correcting) parsing and similarity measure in the next section. The distance measures defined in this chapter are not metric. They have the properties of positivity and symmetry, but do not necessarily have the property of triangle inequality. The accuracy of actual similarity

measure depends on many parameters. The most significant one is the assignment of weights and probabilities. The weights and probabilities assignment is case-dependent and usually heuristic. Previous knowledges and statistics may guide the assignment in some cases.

## 2.3 Error-Correcting Parsing

Error-correcting parser (ECP) has been proposed in the areas of compiler design (Aho and Peterson, 1972) and syntactic pattern recognition (Fu, 1977). When a conventional parser fails to parse a string, it will terminate and reject the string. An error-correcting parser produces same results as a conventional one when the string is syntactically correct. However, it also generates a parse for the string even when it has minor syntax errors. The significance of error-correcting parsing in compiler design is still controversial since it may misinterprete the programmer's intention. However, its significance in syntactic pattern recognition is unquestionable. The most important reason is the noise problem. The noise may come from sensor device, environment or data communication. These will cause segmentation error and primitive recognition error, and therefore result in syntax error In many cases, the pattern grammars are constructed from a finite set of training samples, and then used to recognize a larger set of test samples. Therefore, it is not surprising that the conventional parsers usually fail to work.

The error-correcting parsing algorithms can be classified into two categories, one uses minimum-distance criterion the other uses maximum-likelihood criterion. The minimum-distance error-correcting

parser (MDECP) is for nonstochastic models where string similarity is measured by distance. The maximum-likelihood error-correcting parser (MLECP) is for stochastic model where string similarity is measured by probability.

The ECP in this chapter is different from other existing ECP's in two aspects; first, it uses symmetric similarity measures, second, it does not use expanded grammar.

### 2.3.1 Minimum-Distance Error-Correcting Parsing Algorithm

For the purpose of generality we will discuss context-free grammar (CFG) throughout this chapter. Since finite-state laguage (FSL) is a subset of context-free language, all the principles described here can be applied to FSL as well. Of course, the implementation can be modified to increase the efficiency. Given a CFG $G$ and an input string $y \in \Sigma^*$, a minimum-distance error-correcting parser (MDECP) generates a parse for some string $x \in L(G)$ such that the distance between $x$ and $y$, $d(x,y)$ is as small as possible. Since we have defined several different string distance, therefore different error-correcting parsers can be constructed.

Aho and Peterson (1972) have shown a minimum-distance error-correcting parsing algorithm which uses the Levenshtein distance. We will call their algorithm "Algorithm A" for short. They first transformed the original grammar into an expanded grammar which includes all the possible error productions. Then, they modified the Earley's parsing algorithm so that the number of error productions used is stored in the item list. The productions of the expanded grammar, $P'$, is constructed from $P$ as follows:

(1) For each production in $P$, replace all terminals $a \in \Sigma$ by by a new nonterminal $E_a$ and add these productions to $P'$.

(2) Add to $P'$ the productions

a) $S' \rightarrow S$

b) $S' \rightarrow SH$

c) $H \rightarrow HI$

d) $H \rightarrow I$

(3) For each $a \in \Sigma$, add to $P'$ the productions

a) $E_a \rightarrow a$

b) $E_a \rightarrow b$ for all $b$ in $\Sigma$, $b \neq a$

c) $E_a \rightarrow Ha$

d) $I \rightarrow a$

e) $E_a \rightarrow \lambda$, $\lambda$ is the empty string

In step (3), the productions $E_a \rightarrow b$, $I \rightarrow a$ and $E_a \rightarrow \lambda$ are called terminal error productions. The production $E_a \rightarrow b$ introduces a substituition error. $I \rightarrow a$ intorduces an insertion error. $E_a \rightarrow \lambda$ introduces a deletion error. For the Levenshtein distance, a constant weight, e.g., 1, is associated with each of these productions. It will also handle the type 1 WLD $d^{W1}(x,y)$ and type 2 WLD $d^{W2}(x,y)$ in a similar way. For the type 1 WLD, weight $\sigma$ is associated with production $E_a \rightarrow$ b, weight $\gamma$ with $E_a \rightarrow \lambda$ and weight $\rho$ with $I \rightarrow a$. For the type 2 WLD, weight $S(a,b)$ is associated with production $E_a \rightarrow b$, weight $D(a)$ with $E_a \rightarrow \lambda$ and weight $I(a)$ with $I \rightarrow a$. However, the problem will occur when it comes to type 3 WLD $d^{W3}(x,y)$. In order to maintain the symmetric property we must have $D(a,b) = I(a,b)$ for all $b \in \Sigma$, $a \in \Sigma'$ as mentioned in Theorem 2.3. The expanded grammar will have difficulty in handling context-dependent transformation weight.

Although we can modify this expanded grammar to handle insertion weights, as did in Fu (1982), it still can not handle the deletion weights. Since the productions associated with context-dependent deletion weights will be something like $bE_a \rightarrow E_a$, $D(a,b)$, but this is not a context-free production rule, even not a context-sensitive production rule. While the expanded grammars seem unable to solve the symmetric problem, we can implement the ECP without the expanded grammar. This idea of ECP without expanded grammar has appeared in Lyon (1974) where type 0 distance is used. His main concern is for practical reasons: to save space and execution time. Our proposed ECP using type 3 WLD is a modified Earley's parsing algorithm where the substitution, insertion and deletion transformations are examined during the parsing.

*Algorithm 2.4.* Minimum-Distance Error-Correcting Parsing Algorithm

*Input:* A grammar $G = (N,\Sigma,P,S)$, an input string $y = b_1 b_2 \ldots b_m$ in $\Sigma^*$, and the weights of transformations between symbols.

*Output:* The parse lists $I_0, I_1, \ldots, I_m$, and $d(x,y)$ where $x$ is the minimum-distance correction of $y$, $x \in L(G)$.

*Method:*

(1) Set $j = 0$. Add $[S \rightarrow \cdot\, \alpha,0,0]$ to $I_j$ if $S \rightarrow \alpha$ is a production in $P$.

(2) Repeat step (3) and (4) until no new items can be added to $I_j$.

(3) If $[A \rightarrow \alpha \cdot B\beta,i,\xi]$ is in $I_j$,and $B \rightarrow \gamma$ is a production in $P$. then add item $[B \rightarrow \cdot\, \gamma,j,0]$ to $I_j$.

(4) If $[A \rightarrow \alpha \cdot\, ,i,\xi]$ is in $I_j$ and $[B \rightarrow \beta \cdot A\gamma,k,\zeta]$ is in $I_i$, and if no item of the form $[B \rightarrow \beta A \cdot \gamma,k,\varphi]$ can be found in $I_j$, then add an item

$[B \rightarrow \beta A \cdot \gamma, k, \xi+\zeta]$ to $I_j$. Store with this item two pointers. The first points to item $[B \rightarrow \beta \cdot A\gamma, k, \zeta]$ in $I_i$; the second points to item $[A \rightarrow \alpha \cdot, i, \xi]$ in $I_j$. If $[B \rightarrow \beta A \cdot \gamma, k, \varphi]$ is already in $I_j$, then replace $\varphi$ by $\xi+\zeta$ together with the pointers if $\varphi > \xi+\zeta$.

(5) For each $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_j$, add $[B \rightarrow \alpha a \cdot \beta, i, \xi + D(b_j, a)]$ to $I_j$. Store with this item a pointer to item $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_j$. If no more new item of this form can be found, go to step (6); otherwise, go to step (2).

(6) If $j = m$, go to step (9); otherwise $j = j + 1$.

(7) For each item $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_{j-1}$ add $[B \rightarrow \alpha a \cdot \beta, i, \xi + S(a, b_j)]$ to $I_j$. Store with this item a pointer to item $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_{j-1}$.

(8) For each item $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_{j-1}$ add $[B \rightarrow \alpha \cdot a\beta, i, \xi + I(a, b_j)]$ to $I_j$. Store with this item a pointer to item $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_{j-1}$. Go to (2).

(9) If item $[S \rightarrow \alpha \cdot, 0, \xi]$ is in $I_m$, then $d(x, y) = \xi$. If there are more than one such items, then choose one with the smallest $\xi$. Exit.

In this algorithm, step (5) examines deletion transformations, step (7) examines substitution transformations and step (8) examines insertion transformations.

The right parse of the input string can be constructed from the parse lists. Since we use error-correcting parsing, it is possible that there may exist several parses associated with one input string, but we only choose the one with minimum distance.

*Algorithm 2.5.* Construction of a right parse from the parse lists

*Input:* $I_0, I_1, \ldots, I_m$, the parse lists for string $y = b_1 b_2 \cdots b_m$

*Output:* A parse $\pi$ for $x$, $x \in L(G)$, and the distance

$d^{W3}(x,y)$ is minimum among all the strings in $L(G)$.

*Method:*

(1) In $I_m$ choose an item of the form $[S \rightarrow \alpha \cdot ,0,\xi]$ where $\xi$ is as small as possible.

(2) Let $\pi$ be the empty string initially, and then execute the routine $R([S \rightarrow \alpha \cdot ,0,\xi],m)$ where $R([A \rightarrow \alpha \cdot \beta,i,\eta],j)$ is defined as follows:

*a*) If $\beta=\lambda$, then let $\pi$ be the previous value of $\pi$ followed by the production number of $A \rightarrow \alpha$. Otherwise, $\pi$ is unchanged.

*b*) If $[A \rightarrow \alpha \cdot \beta,i,\eta]$ has only one pointer, then execute the item where it points to. It may be $R([A \rightarrow \alpha \cdot \beta,i,\zeta],j-1)$, $R([A \rightarrow \alpha' \cdot a\beta,i,\zeta],j-1)$ or $R([A \rightarrow \alpha' \cdot a\beta,i,\zeta],j)$ where $\alpha=\alpha' a$. Return.

*c*) If $[A \rightarrow \alpha \cdot \beta,i,\eta]$ has two pointers and $\alpha=\alpha' B$, then execute $R([B \rightarrow \gamma \cdot ,h,\mu],j)$ followed by $R([A \rightarrow \alpha' \cdot B\beta,i,\psi],h)$. Return.

*d*) If $\alpha=\lambda$, return.

The parse constructed by Algorithm 2.5 is for $x$, $x \in L(G)$, i.e., no error productions are included. Usually there is no need to know the error productions (or equivalently error transformations); but if we do need to know, we can store the information like $D(b_j,a)$, $S(a,b_j)$ or $I(a,b_j)$ in each item. Then we can extract the exact transformations when we execute $R$ routines. If we are only interested in the minimum distance, for example, to determine the class membership, then Algorithm 2.4 will be sufficient.

Algorithm 2.4 is more powerful (because its parse is in terms of symmetric distance) and is at least as efficient as Algorithm A.

*Lemma* 2.1: The time complexity of Algorithm 2.4 is $O(n^3)$ where $n$ is the length of the input string.

The proof of lemma 2.1 is similar to that of (Aho and Peterson, 1972). Since each item list $I_j$ takes time $O(j^2)$ to complete, therefore the total time is $O(n^3)$. We can also show that the number of productions and the number of items in item lists of Algorithm 2.4 are less than those of Algorithm A. Therefore, less numbers of productions and items have to be considered when we add new items to item lists. For each item $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_{j-1}$ in Algorithm 2.4 there is an item $[B \rightarrow \alpha \cdot E_a\beta, i, \xi]$ in $I_{j-1}$ in Algorithm A. Let us consider the following transformations:

(1) ***Substitition***. There is an item $[E_a \rightarrow \cdot\, b, j-1, S(a,b)]$ in $I_{j-1}$ where $b=b_j$ and $[E_a \rightarrow b \cdot, j-1, S(a,b)]$, $[B \rightarrow \alpha E_a \cdot \beta, i, \xi+S(a,b)]$ in $I_j$ in Algorithm A. There is only one item $[B \rightarrow \alpha a \cdot \beta, i, \xi+S(a,b_j)]$ in $I_j$ in Algorithm 2.4.

(2) ***Deletion***. There is an item $[E_a \rightarrow \cdot\, \lambda, j-1, D(a)]$ and $[B \rightarrow \alpha E_a \cdot \beta, i, \xi+D(a)]$ in $I_{j-1}$ in Algorithm A. There is only one item $[B \rightarrow \alpha a \cdot \beta, i, \xi+D(b_j,a)]$ in $I_{j-1}$ in algorithm 2.4.

(3) ***Insertion***. There are items $[E_a \rightarrow Ha, j-1, 0]$, $[H \rightarrow \cdot\, I, j-1, 0]$ and $[I \rightarrow \cdot\, b, j-1, I(b)]$ where $b=b_j$ in $I_{j-1}$ and items $[I \rightarrow b \cdot, j-1, I(b)]$, $[H \rightarrow I \cdot, j-1, I(b)]$ and $[E_a \rightarrow H \cdot a, j-1, I(b)]$ in $I_j$ in Algorithm A. There is only one item $[B \rightarrow \alpha \cdot a\beta, i, \xi+I(a,b_j)]$ in $I_j$ in Algorithm 2.4.

Since all the other items not involving error transformations are unchanged, therefore we can see that the time complexity of Algorithm 2.4 is no more than that of Algorithm A, i.e., the time complexity of Algorithm 2.4 is $O(n^3)$.

We have shown a minimum-distance error-correcting parsing algorithm for any nonstochastic CFG. The distance is symmetric and can be any one described in Section 2.2. For a stochastic CFG, we can also construct a maximum-likelihood error-correcting parser which will be discussed in the next section.

### 2.3.2 Maximum-Likelihood Error-Correcting Parsing Algorithm

Given a stochastic context-free grammar (SCFG) $G_s$ and an input string $y \in \Sigma^*$, a maximum-likelihood error-correcting parser (MLECP) generates a parse for some string $x \in L(G_s)$ such that the probability $p(y|x)p(x)$ is the maximum, where $p(y|x)$ is the deformation probability from string $x$ to $y$ and $p(x)$ is the probability associated with string $x$ in $L(G_s)$ (Fu, 1982). There may exist more than one derivation trees for each $x \in L(G_s)$ unless the grammar $G_s$ is unambiguous. Meanwhile, there will be many possible transformations from string $x$ to $y$. We define $p(y|x)p(x)$ as the one with maximum probability, i.e.,

$$p(y|x)p(x) = \max_{i,j} q_j(y|x)p_i(x)$$

where $p_i(x)$ is the probability associated with the $i$th distinct derivation of string $x$ and $q_j(y|x)$ is the probability associated with the $j$th distinct transformation from $x$ to $y$. The probability $p(y|x)$ which is equal to $\max_j q_j(y|x)$ is exactly the same as what we defined for string similarity in Section 2.4.

The proposed MLECP is a modified Earley's parsing algorithm. It does not require an expanded grammar and is applicable to ambiguous grammars. The transformation probabilities as well as the insertion,

deletion and substitution transformations are examined during the parsing. The partial probabilies are stored in each item list. Pointers to the previous items are also stored in the item lists to save parse extraction time.

*Algorithm 2.6.* Maximum-Likelihood Error-Correcting Parsing Algorithm

*Input:* A stochastic grammar $G_s = (N,\Sigma,P_s,S)$, an input string $y = b_1 b_2 \ldots b_m$ in $\Sigma^*$, and the probabilities of transformations.

*Output:* The parse lists $I_0, I_1,\ldots,I_m$, and $p(y \mid x)p(x)$ where $x$ is the maximum-likelihood correction of $y$, $x \in L(G_s)$.

*Method:*

(1) Set $j = 0$. Add $[S \rightarrow \cdot\, \alpha,0,p]$ to $I_j$ if $S \overset{p}{\rightarrow} \alpha$ is a production in $P$.

(2) Repeat step (3) and (4) until no new items can be added to $I_j$.

(3) If $[A \rightarrow \alpha \cdot B\beta,i,\xi]$ is in $I_j$,and $B \overset{q}{\rightarrow} \gamma$ is a production in $P$, then add item $[B \rightarrow \cdot\, \gamma,j,q]$ to $I_j$.

(4) If $[A \rightarrow \alpha \cdot\, ,i,\xi]$ is in $I_j$ and $[B \rightarrow \beta \cdot A\gamma,k,\zeta]$ is in $I_i$, and if no item of the form $[B \rightarrow \beta A \cdot \gamma,k,\varphi]$ can be found in $I_j$, then add an item $[B \rightarrow \beta A \cdot \gamma,k,\xi \cdot \zeta]$ to $I_j$. Store with this item two pointers. The first points to item $[B \rightarrow \beta \cdot A\gamma,k,\zeta]$ in $I_i$; the second points to item $[A \rightarrow \alpha \cdot\, ,i,\xi]$ in $I_j$. If $[B \rightarrow \beta A \cdot \gamma,k,\varphi]$ is already in $I_j$, then replace $\varphi$ by $\xi \cdot \zeta$ together with the pointers if $\varphi < \xi \cdot \zeta$.

(5) For each $[B \rightarrow \alpha \cdot a\beta,i,\xi]$ in $I_j$, add $[B \rightarrow \alpha a \cdot \beta,i,\xi \cdot P_D(a \mid b_j a)]$ to $I_j$. Store with this item a pointer to item $[B \rightarrow \alpha \cdot a\beta,i,\xi]$ in $I_j$. If no more new item of this form can be found, go to step (6); otherwise, go to step (2).

(6) If $j=m$, go to step (9); otherwise $j=j+1$.

(7) For each item $[B\rightarrow\alpha\cdot a\beta,i,\xi]$ in $I_{j-1}$ add $[B\rightarrow\alpha a\cdot\beta,i,\xi\cdot P_S(b_j\,|\,a)]$ to $I_j$. Store with this item a pointer to item $[B\rightarrow\alpha\cdot a\beta,i,\xi]$ in $I_{j-1}$.

(8) For each item $[B\rightarrow\alpha\cdot a\beta,i,\xi]$ in $I_{j-1}$ add $[B\rightarrow\alpha\cdot a\beta,i,\xi\cdot P_I(b_j a\,|\,a)]$ to $I_j$. Store with this item a pointer to item $[B\rightarrow\alpha\cdot a\beta,i,\xi]$ in $I_{j-1}$. Go to (2).

(9) If item $[S\rightarrow\alpha\cdot,0,\xi]$ is in $I_m$, then $p(y\,|\,x)p(x)=\xi$. If there are more than one such items, then choose one with the largest $\xi$. Exit.

The right parse can be extracted from the parse lists. Algorithm 2.5 can be applied here except that in step (1) we choose an item of the form $[S\rightarrow\alpha\cdot,0,\xi]$ in $I_m$ which is as large as possible. The parse extracted here contains no error productions. We can also store and extract the error transformations as did in the last section. The time complexity of Algorithm 2.6 is also $O(n^3)$ since the procedures are almost identical to those of Algorithm 2.4.

*Lemma* 2.2: The time complexity of Algorithm 2.6 is $O(n^3)$ where $n$ is the length of the input string.

Suppose $G_s'$ is an expanded grammar, then the stochastic language generated by $G_s'$ is

$$L(G_s')=\left\{(y,p(y))\;|\;y\in\Sigma^*,p(y)=\sum_{x\in L(G_s)}\sum_{i=1}^{r}q_i(y\,|\,x)p(x)\right\}$$

where $r$ is the number of distinct transformations from string $x$ to $y$, $q_i(y\,|\,x)$ is the probability associated with the $i^{th}$ transformation and $p(x)$ is the probability associated with $x$. Although string $y$ is

generated by the expanded grmmar $G_s'$, the probability associated with $y$, $p(y)$, can be computed without the expanded grammar.

*Algorithm 2.7.* Computation of String Probability

*Input:* A stochastic grammar $G_s = (N,\Sigma,P_s,S)$, an input string $y = b_1b_2...b_m$ in $\Sigma^*$, and the probabilities of transformations.

*Output:* The probability associated with $y$, $p(y)$, where $y$ is generated by the expanded grammar $G_s'$.

*Method:*

(1) Set $j = 0$. Add $[S \rightarrow \cdot\ \alpha,0,p]$ to $I_j$ if $S \overset{p}{\rightarrow} \alpha$ is a production in $P$.

(2) Repeat step (3) and (4) until no new items can be added to $I_j$.

(3) If $[A \rightarrow \alpha \cdot B\beta,i,\xi]$ is in $I_j$,and $B \overset{q}{\rightarrow} \gamma$ is a production in $P$, then add item $[B \rightarrow \cdot\ \gamma,j,q]$ to $I_j$.

(4) If $[A \rightarrow \alpha \cdot\ ,i,\xi]$ is in $I_j$ and $[B \rightarrow \beta \cdot A\gamma,k,\zeta]$ is in $I_i$, and if no item of the form $[B \rightarrow \beta A \cdot \gamma,k,\varphi]$ can be found in $I_j$, then add an item $[B \rightarrow \beta A \cdot \gamma,k,\xi\cdot\zeta]$ to $I_j$. If $[B \rightarrow \beta A \cdot \gamma,k,\varphi]$ is already in $I_j$, then replace $\varphi$ by $\varphi+\xi\cdot\zeta$.

(5) For each $[B \rightarrow \alpha \cdot a\beta,i,\xi]$ in $I_j$, add $[B \rightarrow \alpha a \cdot \beta,i,\xi\cdot P_D(a \mid b_j a)]$ to $I_j$. If no more new item of this form can be found, go to step (6); otherwise, go to step (2).

(6) If $j=m$, go to step (9); otherwise $j=j+1$.

(7) For each item $[B \rightarrow \alpha \cdot a\beta,i,\xi]$ in $I_{j-1}$ add $[B \rightarrow \alpha a \cdot \beta,i,\xi\cdot P_S(b_j \mid a)]$ to $I_j$.

(8) For each item $[B \rightarrow \alpha \cdot a\beta, i, \xi]$ in $I_{j-1}$ add $[B \rightarrow \alpha \cdot a\beta, i, \xi \cdot P_l(b_j a \mid a)]$ to $I_j$. Go to (2).

(9) For all items of the form $[S \rightarrow \alpha_i \cdot, 0, \xi_i]$ is in $I_m$, sum up all the $\xi_i's$. $p(y) = \sum_i \xi_i$. Exit.

Algorithm 2.7 is useful in computing the class conditional probability which is used in Bayes' decision rule. Given a string $y$, Algorithm 2.7 is able to compute the probability that $y$ is generated by $G_s'$, i.e., $p(y \mid G_s')$. Even for a string $x \in L(G_s)$, we can compute the probability $p(x \mid G_s)$ where $G_s$ is the original stochastic grammar. This capability is important since from generation point of view it is very difficult to find the summation of the probabilities of all the possible derivations. But if we go the other way, i.e., by parsing, it is very easy to get the probability. If we are dealing with $G_s$ only, i.e., to find the probability $p(x \mid G_s)$, $x \in L(G_s)$, then we should skip step (5), (7) and (8) of Algorithm 2.7. Since these steps are for deletion, substitution and insertion deformations.

## 2.4 Recognition Procedures for Syntactic Patterns

In syntactic pattern recognition, if classification is the only purpose, then we can use either ECP or NNR. Given a CFG $G$ and an input string $y \in \Sigma^*$, a MDECP generates a parse for some string $x \in L(G)$ such that the distance between $x$ and $y$ is as small as possible. This distance is defined as the distance between string $y$ and set $L(G)$. On the other hand, a MLECP generates a parse for some string $x \in L(G_s)$ such that the probability $p(y \mid x)p(x)$ is the maximum. This probability

is defined as the likelihood that string $y$ belongs to set $L(G_s)$. The nearest-neighbor rule for the case that $G$ is nonstochastic computes the distances between $y$ and all the string $x$, $x \in L(G)$, and select the one corresponding to the smallest distance. The nearest-neighbor rule for stochastic case computes the probabilities $p(y \mid x)$ for all $x \in L(G_s)$ and select the one such that the product $p(y \mid x)p(x)$ is the maximum. The probability density function for $x$, $p(x)$, is assumed known.

If $L(G)$ is finite then either MDECP or NNR can be used to find $x$ and $d(x,y)$. Similarly, if $L(G_s)$ is finite then we can use either MLECP or NNR to find $x$ and $p(y \mid x)p(x)$. The results of ECP and NNR may be different depending on how the grammar $G$ and $G_s$ are constructed. If $L(G)$ or $L(G_s)$ is not finite, then the NNR will not be able to test the whole $L(G)$ or $L(G_s)$. Thus, it is necessary to find a finite subset of $L(G)$ or $L(G_s)$ so that the NNR can be implemented. This is also true even when $L(G)$ or $L(G_s)$ is finite but with a size hard to manage. However, neither MDECP nor MLECP has difficulty in dealing with infinite language. Therefore, it is advantageous to use MDECP or MLECP when $L(G)$ or $L(G_s)$ is infinite, since the recognition accurancy of NNR may be degraded because of the limited size of prototypes. But in real application, we usually encounter a finite set of samples and need to construct or infer a grammar from these samples. In this case, the recognition results of these two approaches will be equal if the constructed or inferred grammar generates exactly the original samples. Therefore, the only factor affecting the selection of algorithm is computation speed. The NNR uses dynamic programming technique whose time complexity is $O(n^2)$ where $n$ is the length of input string. The complexity of MDECP and MLECP is also $O(n^2)$ if the grammar is

unambiguous. Although both ECP and NNR have $O(n^2)$ time complexity, NNR is usually faster than ECP. We will see an example in chapter III.

## 2.5 Conclusion

We have discussed four types of string similarity measures in this chapter, and the conditions for them to be symmetric. We also proposed parsing algorithms to deal with the symmetric problem which can not be carried out by any other ECP. These algorithms are at least as efficient (computation-wise) as other parsing algorithms. A minimum-diatance criterion is used for nonstochastic models and a maximum-likelihood criterion is used for stochastic models for both ECP and NNR. Bayes' decision rule can be applied when dealing with multiclass problems of stochastic models. The class conditional probability $p(x \mid C_i)$, where $C_i = L(G_i)$, can be computed by Algorithm 2.6.

In NNR, the distance computation employs a dynamic programming procedure which makes it very easy for implementation in VLSI architectures. VLSI architectures for ECP and string distances computation will be reviewed in Chapter V. We also propose a VLSI architecture for computing the string (Levenshtein) distance in Chapter V.

# CHAPTER III

# APPLICATIONS OF SYNTACTIC PATTERN RECOGNITION TO SEISMIC CLASSIFICATION

## 3.1 Introduction

In this chapter we apply syntactic approaches to two real seismic classification problems. One is the seismic discrimination between nuclear explosion and natural earthquake, the other is the seismic classification in structural damage assesment. These waveforms have been sampled and digitized before we obtain the data. However, various noises exist in both cases. Certain prepocessing procedures therefore must be imposed to remove those noises. Section 2 to 5 discuss application to seismic discrimination, and Section 6 shows application to damage assesment.

Seismological methods are so far the most effective and practical methods for detecting nuclear explosions, especially for underground explosions. Position, depth and origin time of the seismic events are useful information for discrimination; so are the body wave magnitude and surface wave magnitude of the seismic wave (Bolt, 1976; Dahlman and Israelson, 1977). Unfortunately, they are not always applicable and reliable for small events. It would be very helpful if the discrimination is based on the short-period waves alone. The application of pattern recognition techniques to seismic wave analysis has been studied

extensively in the last few years (Chen, 1978; Tjostheim, 1978; Sarna and Stark, 1980). They all use short-period waves only for discrimination. Most of these studies concentrated on feature selection. Only simple decision-theoretic techniques have been used. However, syntactic pattern recognition appears to be quite promising in this area. It uses the structural information of the seismic wave which is very important in analysis. Seismic records are one-dimensional waveforms. Although there exist several alternatives (Ehrich and Foith, 1976; Sankar and Rosenfeld, 1979) for representing one-dimensional waveforms, it is most natural to represent them by sentences, i.e., strings of primitives. In order to make it easy for analysis we divide the pattern representation procedure into three steps, namely, pattern segmentation, feature selection and primitive recognition, though they are correlated.

In this chapter, we apply two different methods of syntactic approach to the recognition of seismic waves. One uses the nearest-neighbor decision rule, the other uses the error-correcting parsing. In the first method, a pattern representation sybsystem converts the seismic waveforms into strings of primitives. The string-to-string distances between the test sample and all the training samples are computed and then the nearest-neighbor decision rule is applied. The second method contains pattern representation, automatic grammatical inference and error-correcting parsing. The pattern representation subsystem performs pattern segmentation, feature selection and primitive recognition so as to convert the seismic wave into a string of primitives. The automatic grammatical inference subsystem infers a finite-state (regular) grammar from a finite set of training samples. The

error-correcting parser performs syntax analysis and classification. Human interaction is required only at the training stage, mostly in pattern representation and slightly in grammatical inference.

## 3.2 Preprocessing

The two major problems in preprocessing of digital signal is to identify the appropriate portion for recognition and to eliminate noise. For example, the voiced portion should be seperated from the unvoiced portion in speech recognition; each ECG cycle should be determined in ECG analysis, and the 'signal' should be recognized in seismic analysis. We will not discuss these in any detail, though they are important. The main reason is the variety of their characters. The seismic signals in our experiment were selected from a huge seismic database. They all have equal length and have been aligned at the onset.

Noise is always a major problem in digital signal processing. Filtering is the most common technique to remove noise, high-pass, low-pass, band-pass, just to name a few. These filters eliminate certain regions of frequency component. Sometimes this may not be desired. For example, in Figure 3.1, there is a pulse-like noise within the seismic signal. This kind of noise is sometimes called glitch. If we apply the signal through a low-pass filter, it can not eliminate the pulse completely, meanwhile all the high frequency components of the signal will also be eliminated. This is not what we want. To avoid this, we need a local filter which will remove only the pulse noise and leave the rest of the signal unchanged. This local filtering is possible because the normal signal does not have pulse in it, the local filter can detect the pulses

and then remove them. This local filtering needs human interaction to specify threshold. Different regions need different thresholds. We can see from Figure 3.1 that the whole signal can be divided into three portions. The relatively flat portion at the beginning is the background noise, which should not be confused with the noise we want to eliminate. The next portion has the strongest signal which is called the signal portion. After the strong signal portion is the weak signal portion which is called coda. A point $i$ is said to be a pulse noise if and only if it satisfies the following two conditions:

(1) absolute magnitude of point $i$, $|a(i)|$, is greater than or equal to the threshold.

(2) absolute value of $a(i+1) + a(i-1) - 2 * a(i)$ is greater than or equal to the threshold.

The second condition seperates the pulse noise from strong signal portion since the pulse noise is much sharper. After point $i$ is detected to be a pulse noise, it can be eliminated by letting

$$a(i) = (a(j) + a(k)) / 2$$

where $j < i$, $k > i$, point $j$ and $k$ are not pulse noise and no point between $j$ and $k$ is normal signal point.

Figure 3.1(a) is a signal before filtering, (b) is the same signal after filtering. Figure 3.2 is another example, but it has more than one pulse noise. From these two examples we can see the local filter works successfully in eliminating the local pulse noise while retaining the original signals.

Another noise problem of seismic signal is the drift during recording. As can be seen from Figure 3.3(b), the whole signal is somewhat

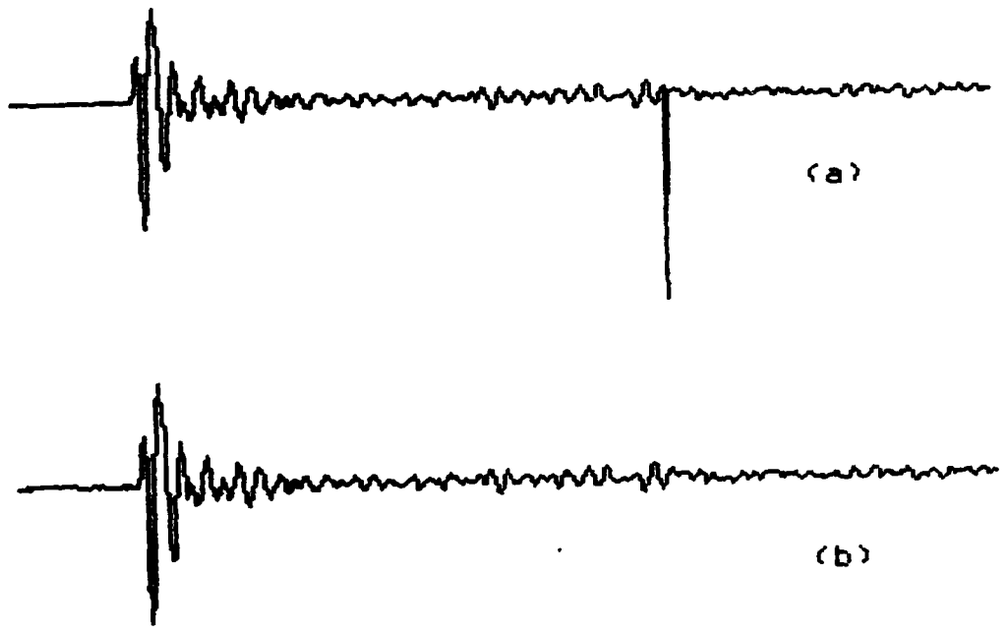



Figure 3.1 (a) An example of seismic signal with pulse noise (glitch). (b) The same waveform after local filtering.

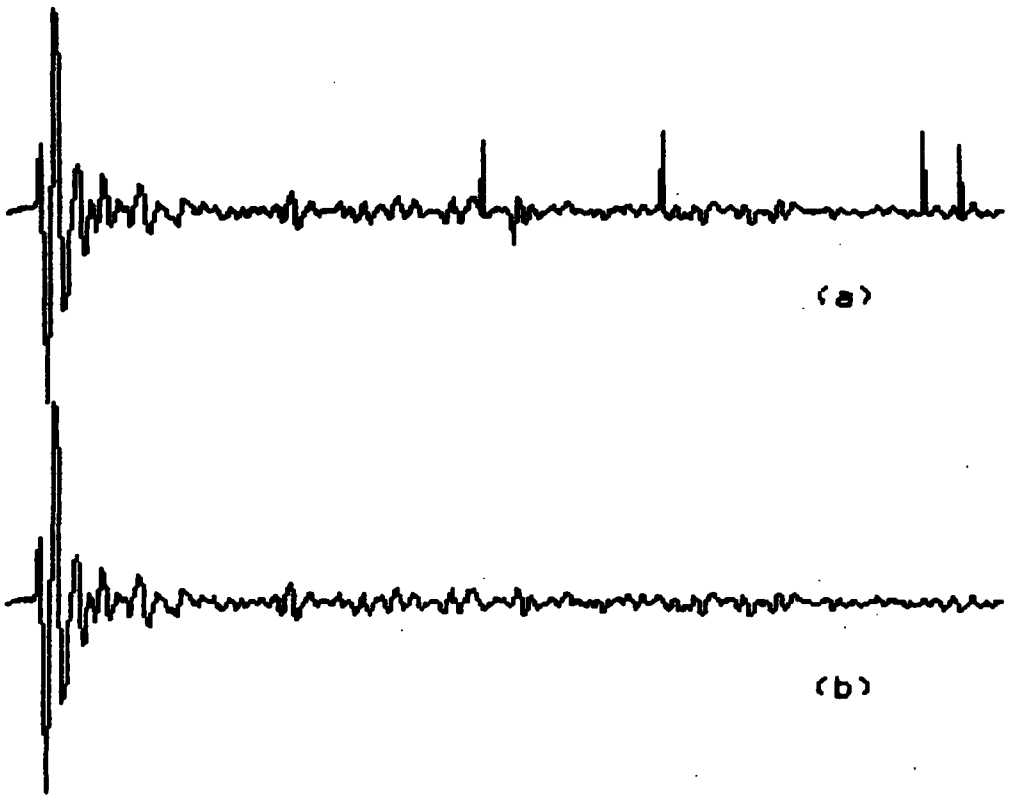



Figure 3.2 (a) Another example of seismic signal with several pulse noise (glitches). (b) The same waveform after local filtering.

below the zero line, especially the beginning portion which is far below the zero line. In order to retain the details of the original signal, we use a low order polynomial regression of the original signal and then subtract this polynomial regression from the original signal. The fitness of the regression is tested by least-squares criterion. We use a 5th-order polynomial regression for the seismic signals. The regression program is taken from the book by Carnahan, Luther and Wilkes (1969). The entire procedure consists of two parts, i.e., global adjustment and local adjustment. In global adjustment, the polynomial regresssion is applied to the whole signal and then followed by subtraction. Figure 3.3(c) is the result after the regression and subtraction from Figure 3.3(b). We can see that the small segment at the beginning still drifts from the zero line slightly. Then we apply regression and subtraction to this small segment; this is called local adjustment. The result after local adjustment is shown in Figure 3.3(d). Another example is shown in Figure 3.4. Figure 3.4(a) is the original signal, (b) is the original signal with the zero line. We can see that the first portion of this signal drift above the zero line and the rest of the signal drifts below the zero line. Figure 3.4(c) is the result after global adjustment and (d) is the result after local adjustment. The sequence of applying global adjustment first and then local adjustment is important. If we reverse the order, it will not produce the same result as we have otherwise. In our present experiment the segment for local adjustment is selected manually. One alternative is to use piece-wise regression to select the optimal breaking point. This is carried out by breaking the whole signal into two segments and then finding regression of each segment. The breaking point which results in miminum deviation is the optimal breaking point. This

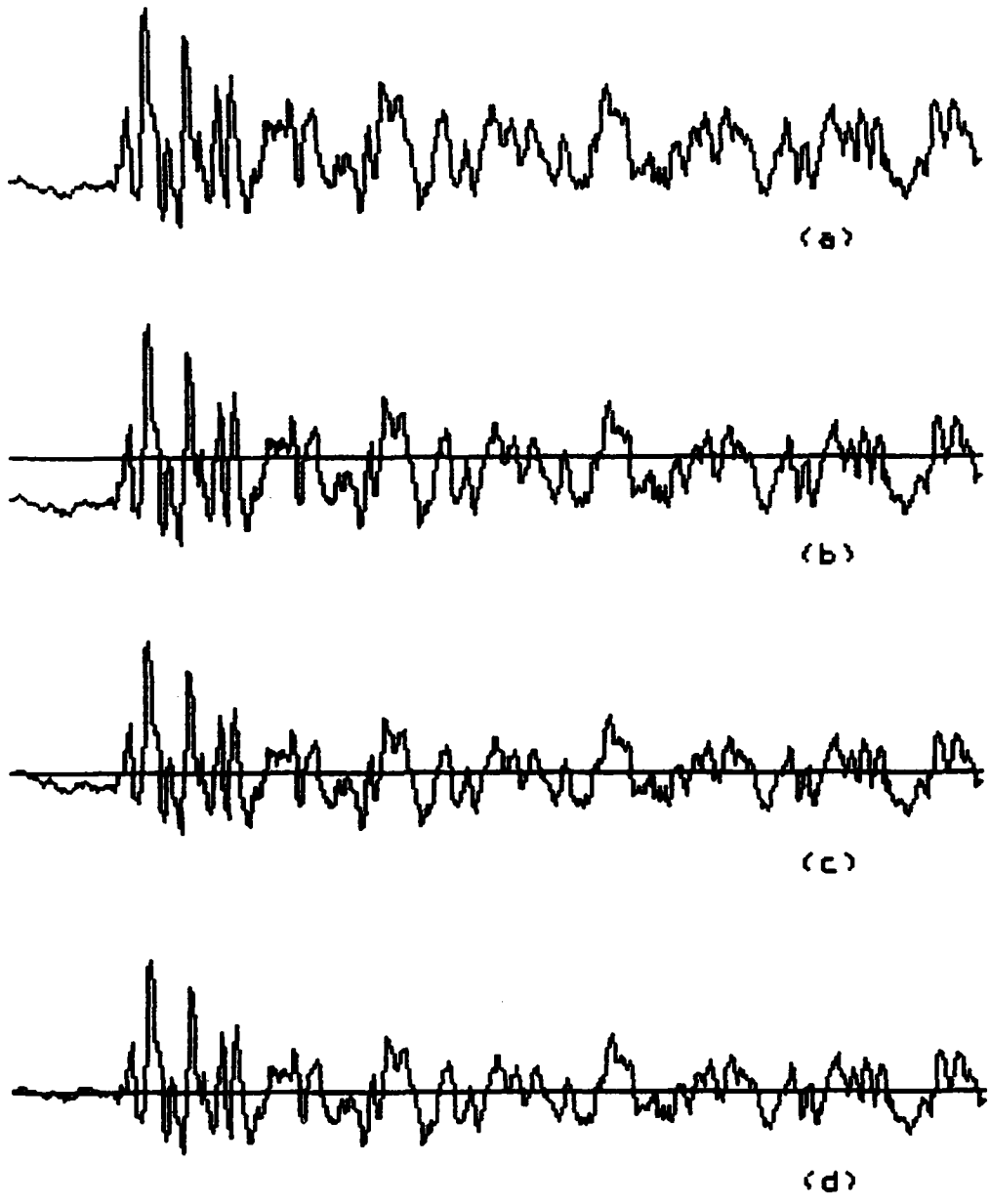



Figure 3.3 (a) An original seismic signal. (b) With zero-line added for comparison. (c) After global adjustment. (d) After local adjustment.

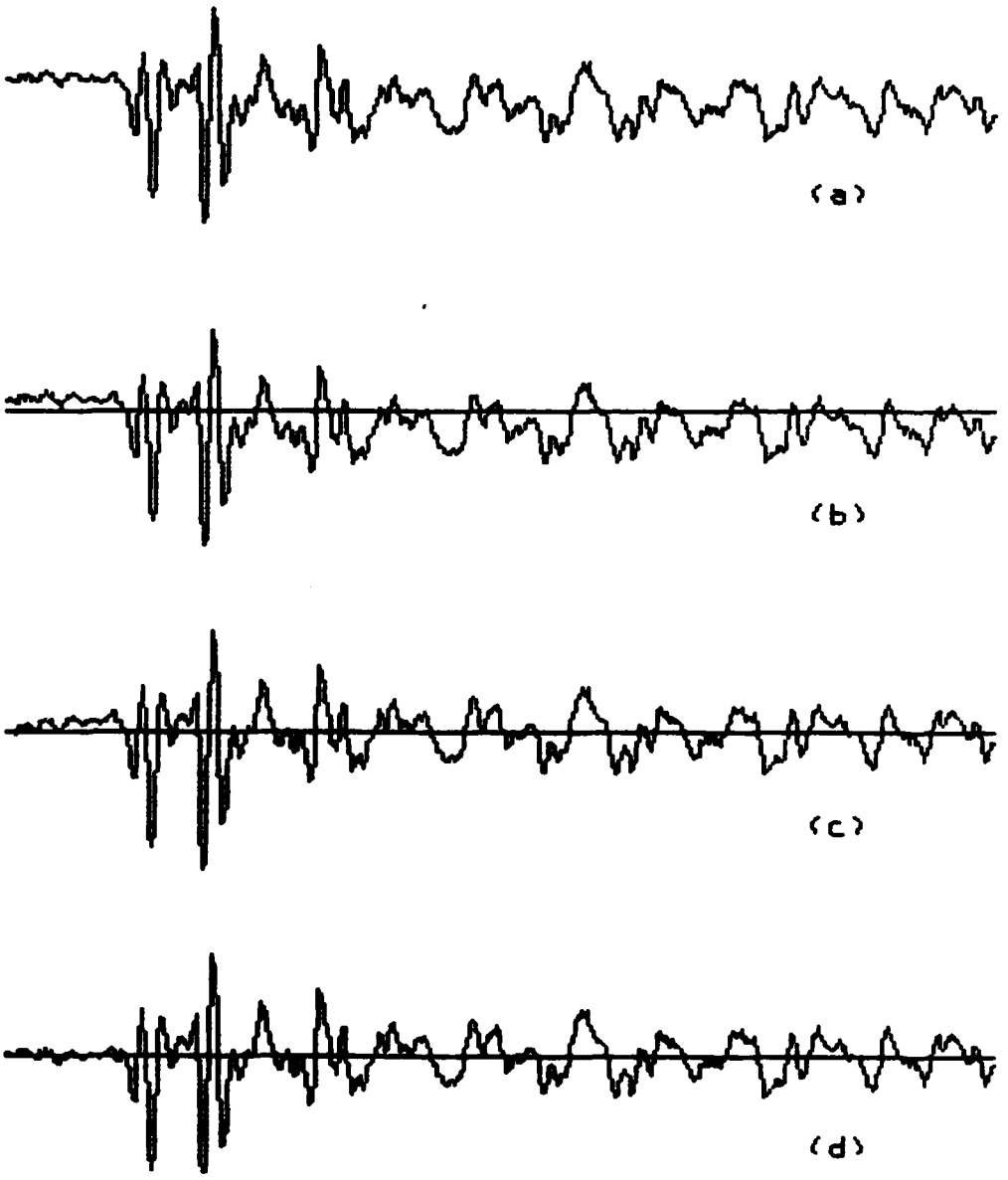



Figure 3.4 (a) Another example of seismic signal. (b) Zero-line is added for comparison. (c) After global adjustment. (d) After local adjustment.

must be done on a section of contiguous points. It is time consuming and therefore is excluded from our experiment. After the above preprocessing procedures we can perform segmentation and primitive selection.

## 3.3 Automatic Clustering Procedure for Primitive Selection

It has been mentioned in Fu (1982) that the pattern primitives should serve as basic pattern elements in describing the structural relations and they should be easily extractable, usually by nonsyntactic methods. The selection of primitives depends largely on the type of waveforms. In some applications, the primitives are prespecified by human expert, e.g., in Giese, et al. (1979). We would like to investigate the possibility of nonsupervised learning in primitive selection, therefore, we use an automatic clustering procedure to select the pattern primitives. This is important because human selection of pattern primitive may not always be available, besides, it may be unreliable.

### 3.3.1 Pattern Segmentation

A digitized waveform to be processed by a digital computer is usually sampled from a continuous waveform which represents the phenomena of a source plus external noise. For some cases, such as ECG and carotid pulse wave analysis (Horowitz, 1975; Stockman, et al., 1976), every single peak and valley are significant, therefore these waveforms can be segmented according to the shape. For others, like EEG (Giese, et al., 1979) and seismic wave analysis in our case, a single

peak or valley does not contain significant information, especially when the signal to noise ratio is low, therefore they should be segmented by length, either a fixed length or a variable length. A variable-length segmentation is more efficient and precise in representation, but it is usually very difficult and time consuming to find an appropriate segmentation. A fixed-length segmentation is much easier to implement. If the length is well selected it will be adequate to represent the original waveform. There is a compromise between the representation accuracy and analysis efficency. The shorter the segmentation is, the more accurate the representation will be. But the analysis becomes more inefficient since the string is longer and the computation time is proportional to string length. Another problem is the noise. If the segmentation is too short, it will be very sensitive to noise.

Pattern segmentation is closely related to primitive selection. The segment length in speech analysis is 20 milliseconds (DeMori, 1972, 1977), and 1 second in EEG analysis (Giese, et al., 1979). For short-period seismic signal, a segment length of around 6 seconds is a good choice. A segment of this length contains adequate information and has been used by many other researchers (Chen, 1978; Tojstheim, 1975). Since the sampling frequency of our data set is 10 Hz, a 6-second period contains 60 points.

We have done experiments on other segment lengths, they are 40 points and 80 points. We selected 41 explosion records out of 111 and 59 earthquake records out of 210 as training samples. The recognition result for 60-point segment length is 91.0%, i.e., 20 misclassifications out of 221. When we chose 40 points as segment length, according to the primitive selection procedure in Section 3.4 the best selecton for

primitive number is 18. For primitive number 18, the recognition result is 72.9%, i.e., 60 misclassifications out of 221. If we chose primitive number 13 as we did in 60-point segment length, the recognition result is still 72.9%, though the detail of classification is different. When we chose 80 points as segment length, the primitive number selection is 14 and the recognition result is 73.8%, i.e., 58 misclassifications out of 221.

Although this experiment is by no means conclusive, it does show that a segment length of 60 points is an appropriate selection for short-period seismic signal. A shorter segment is too sensitive to noise and a longer segment is too complicated for a primitive. The selection of segment length is ususlly a subjective judegment and depends on the characteristic of the signal waveform.

### 3.3.2 Feature Selection

Any linear or nonlinear mapping of the original measurements can be considered as features provided they have discriminating capability. Both time-domain features and frequency-domain features have been used for seismic discrimination . For example, complexity and autoregressive models are features in time domain; spectral ratio and third moment of frequency are features in frequency domain (Dahlman and Israelson, 1977). Since we segment the seismic wave, complexity and spectral ratio features are implicitely contained in the string structure. Furthermore, the segment may be too short for a model estimation if we use shorter segment. Therefore, we selected a pair of commonly used features, i.e., zero crossing count and log energy of each segment, which are easy to compute and contain significant information. Easy to

compute is a desired property for primitive extraction in syntactic approach. Zero crossing count roughly represents the major frequency component of the signal and log energy indicates the magnitude of the signal. These two features should be able to characterize the signal segment. Other features may also serve as good candidates. An advantage of syntactic approach is that feature selection is simpler since features are extracted from smaller segments, and feature selection is not that critical as is in statistical approach. Since there is no optimal feature selection algorithm, features are usually subjectively selected. Although there are criteria such as between cluster and within cluster scatterness, they have no direct relation to final recognition results. While other features, including $K-L$ expansion, do not show any superiority in recognition results in our preliminary experiments, we will stick to the zero crossing count and log energy.

Since we are experimenting a new approach for seismic discrimination, we do not particularly emphasize feature selection. In fact, simple features like these give favorable result in our experiment. This indicates that syntactic approach utilizes structural information instead of sophisticated feature measurement.

### 3.3.3 Primitive Recognition

The selection of primitives varies very largely in digital signal recognition. Line segments from linear approximation of signals have been used in ECG analysis (Horowitz, 1975, 1977). Parabola and line segment have been used in carotid pulse wave analysis (Stockman, et al., 1976). These primitives are mainly used to describe the shape of the signal waveform. When the shape of the signal waveform is not

important, other types of primitives must be selected. For example in spoken word recognition (DeMori, 1972, 1977), silence interval, stable zone and lines are used as primitives. In EEG analysis (Giese, et al., 1979), a group of seven primitives has been specified and a linear classifier is used to recognize the testing segments. What should we do if the signal on hand is not as predictable as speech signal, nor can we specify the primitives as in EEG analysis. One possible solution is by clustering procedure. A clustering procedure will classify any number of signal segments into certain number of clusters in an optimal way, which means minimization of some criterion function.

If the number of primitives, i.e., the number of clusters, has been selected then any typical clustering technique, e.g., $K$-means algorithm, can find the optimal clustering. Now the difficult part is how to select an appropriate primitive number. For example in EEG analysis, how do we know seven is the best selection. Is there any other better selection? How does the selection of primitive number affect the final recognition results? We will discuss all of these questions in this section.

Without lost of generality we assume that each signal segment is represented by a vector of features $x = [x_1, x_2, ..., x_k]^t$. It is noted that we use decision-theoretic approach for primitive selection. Other representations may also serve the purpose as long as the similarity between signal segments can be computed. If the feature space is isotropic, then the Euclidean distance can be used as a measure of similarity and it is invariant under translation or rotation. However, the invariance can be attained by normalizing the data before clustering.

Suppose we want to partition $n$ samples $x^1, x^2, \ldots, x^n$ into $k$ disjoint subsets $C_1, C_2, \ldots, C_k$. Each subset represents a cluster. The samples in the same cluster are more similar than the samples in different clusters. One typical approach is to define a criterion function that measures the clustering quality of any partition of the samples. Then the problem is to minimize of maximize the criterion function. One of the most well-known criterion function is the sum-of-squared-error criterion (Duda and Hart, 1973). Let $n_i$ be the number of samples in cluster $C_i$ and $m_i$ be the mean of those samples, where

$$m_i = \frac{1}{n_i} \sum_{x \in C_i} x$$

The sum-of-squared-error criterion is defined as

$$J_e = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - m_i||^2$$

Another set of criterion functions are derived from scatter matrice. First, let us introduce some definitions.

Mean vector for $i$th cluster:

$$m_i = \frac{1}{n_i} \sum_{x \in C_i} x$$

Total mean vector:

$$m = \frac{1}{n} \sum_{C} x = \frac{1}{n} \sum_{i=1}^{k} n_i m_i$$

Scatter matrix for $i$th cluster:

$$S_i = \sum_{x \in C_i} (x - m_i)(x - m_i)^t$$

Within-cluster scatter matrix:

$$S_W = \sum_{i=1}^{k} S_i$$

Between-cluster scatter matrix:

$$S_B = \sum_{i=1}^{k} n_i (m_i - m)(m_i - m)^t$$

Total scatter matrix:

$$S_T = \sum_{x \in C} (x - m)(x - m)^t$$

It follows obviously that $S_T = S_W + S_B$

We define the optimal partition as one that minimizes $S_W$ or maximizes $S_B$. In doing so we need a scalar measure of the size of a scatter matrix. The trace of $S_W$ is the simplest measures. Other well-known measures are the determinant of $S_W$ and the trace of $S_W^{-1} S_B$. For the sake of computational simplicity we will only consider the trace of $S_W$ as criterion function. The trace criterion is defined as:

$$tr\ S_W = \sum_{i=1}^{k} tr\ S_i = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - m_i||^2 = J_e$$

which is exactly the same as the sum-of-squared-error criterion. Since $tr\ S_T = tr\ S_B + tr\ S_W$ and $tr\ S_T$ is independent of how the samples are partitioned, therefore minimizing $tr\ S_W$ is equivalent to maximizing $tr\ S_B$. Where

$$tr\ S_B = \sum_{i=1}^{k} n_i\ ||m_i - m||^2$$

If the number of cluster is known, then the $K$-means algorithm can be applied to find a clustering which minimizes the criterion function, i.e., the sum-of-squared-error $J_e$. When the number of clusters is unknown, at least two approaches can be used to determine the optimal cluster number. These two approaches turn out to have similar results in our experiment.

Both approaches use a bottom-up hierarchical clustering procedure. This algorithm repeats the clustering procedure for $k = U$, $k = U - 1$ ,..., $k = L$, where $U$ and $L$ are the specified upper and lower bound respectively. The first approach selects the optimal cluster number by examining how the criterion function $J_e$ changes with $k$. If these $n$ samples are really grouped into $p$ well seperated clusters, then $J_e$ should increase slowly until $k = p$ and then increase much more rapidly thereafter. The algorithm for bottom-up clustering procedure is shown as follows:

*Algorithm 3.1* Bottom-Up Hierarchical Clustering

*Input:* A set of $n$ unclassified samples, an upper bound $U$ and a lower bound $L$.

*Output:* A sequence of optimal clusterings for the number of clusters between $U$ and $L$.

*Method:*

(1) Let $k = U$, $k$ is the number of clusters, and arbitrarily assign cluster membership.

(2) Reassign membership using $K$-means algorithm. If

$k \leq L$, stop.

(3) Find the nearest pair of clusters, say $C_i$ and $C_j$, $i \neq j$.

(4) Merge $C_i$ and $C_j$, delete $C_j$ and decrease $k$ by one,

go to step 2.

The distance between two clusters is defined by

$$d(C_i, C_j) = ||m_i - m_j||$$

where $m_i$, $m_j$ are the mean vectors of clusters $i, j$ respectively.

Just as F-statistics can be used in univariate case to test the significance of group seperation, a pseudo F-statistics (PFS) can be applied in multivariate case provided that a single measurement of similarity between samples, e.g., Euclidean distance, is assumed (Vogel and Wong, 1978). A pseudo F-statistics is defined as:

$$PFS = \frac{tr\ S_B\ (n-k)}{tr\ S_W\ (k-1)}$$

As the number of clusters increases, $tr\ S_B$ will always increase while $tr\ S_W$ will always decrease. However, the PFS value will not monotonically increase due to the effect of $(n-k)\ /\ (k-1)$ which is smaller as $k$ becomes larger. Therefore, there will be a peak of PFS value somewhere in the middle. Since, like F-statistics, the PFS shows the significance of group seperation, therefore a larger PFS value means the clusters are more compact and well seperated. The crietrion here is to select the maximum PFS value; the corresponding cluster number will be optimal. For example, in Figure 3.7, the maximum PFS value appears at cluster number 13, therefore 13 is the optimal selection for cluster number.

## 3.4 Syntax Analysis

If the classification is all we need, then the nearest-neighbor decision rule is preferred because of its computational efficiency. On the other hand, if a complete description of the waveform structure is needed, we have to use parsing (or error-correcting parsing). An error-correcting parser (instead of conventional parser) is required for most practical pattern recognition applications. Since noise and distortion usually cause conventional parsers to fail. It is not unusual that even a noise-free, distortion-free pattern can not be recognized by a conventional parser, since the pattern grammar is often inferred from a small set of training samples.

### 3.4.1 Nearest-Neighbor Decision Rule

The concept of nearest-neighbor decision rule in syntactic approach is similar to that in decision-theoretic approach. The only difference is in distance calculation. Four types of string distances have been discussed in chapter two, and they can be computed using dynamic programming method (e.g., Algorithm 2.1).

### 3.4.2 Error-Correcting Finite-State Parsing

Before parsing can take place we must have a grammar, which can be either heuristically constructed or inferred from a set of training samples. In order to study the learning capability of the syntactic method, we choose the grammatical inference approach.

Phrase structure grammars have been used to describe patterns in syntactic pattern recognition (see Fu, 1982). Each pattern is

represented by a string of primitives which corresponds to a sentence in a language (tree or graph in high dimensional grammars). All strings which belong to the same class are generated by one grammar.

*Grammatical Inference*

A set of sentences $S^+$ is a positive sample of a language $L(G)$, if $S^+ \subset L(G)$. A set of sentences $S^-$ is a negative sample of a language $L(G)$, if $S^- \subset \overline{L(G)}$.

A positive sample $S^+$ of a language $L(G)$ is structurally complete if each production in $G$ is used in the generation of at least one string in $S^+$ (Fu and Booth, 1975).

We assume that the set $S^+$ is structurally complete and $S^+ \subset L(G_D)$, where $G_D$ is the inferred grammar. Theoretically, if $S^+$ is a structurally complete sample of the language $L(G)$ generated by the finite-state grammar $G$ then the canonical grammar $G_C$ can be inferred from $S^+$. A set of derived grammars can be derived from $G_C$. The derived grammars are obtained by partitioning the set of nonterminals of the canonical grammar into equivalence classes. Each nonterminal of the derived grammar corresponds to one block of the partition. Since the number of possible partitions is too large it is infeasible to evaluate all the partitions. Therefore some algorithms such as $k$-tail algorithm (Biermann and Feldman, 1972) has been suggested to reduce the number of derived grammars. These algorithms have one disadvantage. The reduced subset of derived grammars may not contain the source grammar. However, it will be sufficient if we only interest in an estimate of the source grammar. There are at least two situations where a grammatical inference algorithm can be used. In the first case there exists

a source grammar which generates a language and we want to infer the source grammar or automaton based on the observed samples. In the second case the exact nature of the source grammar is unknown, the only information we have are some sentences generated by the source. We assume that the source grammar falls into a particular class and infer a grammar which generates all the training samples, and hopefully will generate some samples belonging to the same class. If a negative sample set is given, the inferred grammar must not generate any sample in the negative sample set. Grammars more complex than finite-state grammars and restricted context-free grammars (in Chomsky hierarchy) can not be inferred efficiently without human interaction. Furthermore, there exists no obvious self-embedding property in seismic waves, finite-state grammars will be sufficient in generating power. Therefore we choose finite-state grammars to describe the seismic waves.

The inference of regular grammars has been studied extensively. The *k*-tail algorithm finds the canonical grammar and then merges the states which are *k*-tail equivalent. This algorithm is adjustable, the value of k controls the size of the inferred grammar. Another algorithm called tail-clustering algorithm (Miclet, 1980) also finds the canonical grammar, but then merges the states which have common tails. The original algorithm is not as flexible as the *k*-tail algorithm, but will infer a grammar which is closer to the source grammar in some cases. We can modify the merge criterion to make it more flexible. Since the grammar is inferred from a small set of training samples, we can only expect that the inferred grammar generates all the training samples and will generate other strings which are similar to the training

samples. The generating power of the inferred grammar relies entirely on the merge procedure. If no merge occurs at all, then the inferred grammar generates exactly the same training set, no more no less. Since all the seismic records have the same length and alignment in our experiment, the sentences representing these signals also have the same length.

*Error-Correcting Parsing*

After a grammar is available, either by automatic inference or by manaul construction, the next step is to design a recognizer which will recognize the patterns generated by the grammar. If the grammar G is finite-state, a deterministic finite-state automaton can be constructed to recognize the strings generated by G.

Segmentation and primitives recognition errors due to noise and distortion usually occur in practice. Conventional parsing algorithms can not handle these situations, therefore, an error-correcting parser must be used (Fu, 1977).

Since all the sentences in our example have the same length, only the substitution error needs to be considered. For each production $A \rightarrow aB$ and $A \rightarrow a$ in the original grammar we add $A \rightarrow bB$ and $A \rightarrow b$ respectively to the covering grammar, where $A,B \in N$, $a,b \in \Sigma$, $b \neq a$, N is a set of nonterminal symbols and $\Sigma$ is a set of terminal symbols. Different weights can be assigned to different error productions, therefore, result in a minimum-cost error-correcting parser. The assignment of weights is a crucial problem. We have used the distance between clusters $a$ and $b$ as the weight for substituting $a$ by $b$ and vise versa. Since a finite-state grammar can be represented by a transition

diagram. Thus, a minimum-cost error-correcting parsing is equivalent to finding a minimum-cost path from the initial state to a final state.

*Algorithm 3.2.* Computation of Minimum-Cost

*Input:* A transition diagram with $n$ nodes numbered 1, 2, ..., $n$, where node 1 is the initial state and node $n$ is a final state, and a cost function $C_{ij}(a)$, for $1 \leq i, j \leq n$, $a \in \Sigma$, with $C_{ij}(a) \geq 0$, for all $i$ and $j$. An input string $s$.

*Output:* $m_{1n}$ the lowest cost of any path from node 1 to node $n$ whose sequence is equal to that of the input string $s$.

*Method:*

(1) Set $k = 1$.

(2) For all $1 \leq j \leq n$, $m_{1j} = \min \{m_{1k} + C_{kj}(b)$, for all $1 \leq k \leq n\}$, where $b$ is the $k$th symbol of input string $s$.

(3) If $k < |s|$, increase $k$ by 1 and go to step (2). If $k = |s|$, go to step (4).

(4) Output $m_{1n}$, which is the lowest cost from node 1 to node $n$ following the move of input string $s$. Stop.

Cost function $C_{ij}(a)$ denotes the cost of moving from state $i$ to state $j$ while the input symbol is '$a$'. $m_{1j}$ is the minimum cost from state 1 to state $j$. The computation time of Algorithm 3.2 is linear, i.e., $O(n)$, where $n$ is the length of the input string. This algorithm is a finite-state parsing algorithm where only substitution error is considered. The production number can be stored with $C_{ij}(a)$, and the parse can be stored with $m_{1j}$.

If insertion and deletion errors are to be considered, then the parser is still similar except that we have to compute and store the information V(T, S, $a$) which is the minimum cost of changing character '$a$' into some string which can change the state of the automaton from state T to S (Wagner, 1974). The inclusion of insertion and deletion errors makes the error correction more complete, but assigning appropriate weights to insertion and deletion error is even more difficult.

### 3.5 Experimental Results on Seismic Discriminat.

The seismic data used in our experiments are provided by Professor C. H. Chen of Southeastern Massachusetts University. The data were recorded at LASA in Montana. Each record contains 1200 points; the sampling frequency is 10 points per second. The original data contains 323 records. Due to some technical problems in data conversion only 321 records were received. Among them 111 records are nuclear explosions and 210 records are earthquakes.

We have selected forty-one earthquake records and fifty-nine explosion records as training samples. Each record is divided into 20 segments where each segment contains 60 points. Two features, i.e., zero-crossing count and log energy, are computed from each segment. Table 3.1 shows the criterion function $J_e$ and its increment from cluster number 16 down to 2, which are the results of applying Algorithm 3.1 to the training segments. We can see that the increment of $J_e$ is small before and until cluster number is equal to 13 and then becomes much larger thereafter. Therefore, we say that 13 is an optimal selection of

TABLE 3.1

**The criterion function, increments of criterion function and the classification results of different cluster number selections**

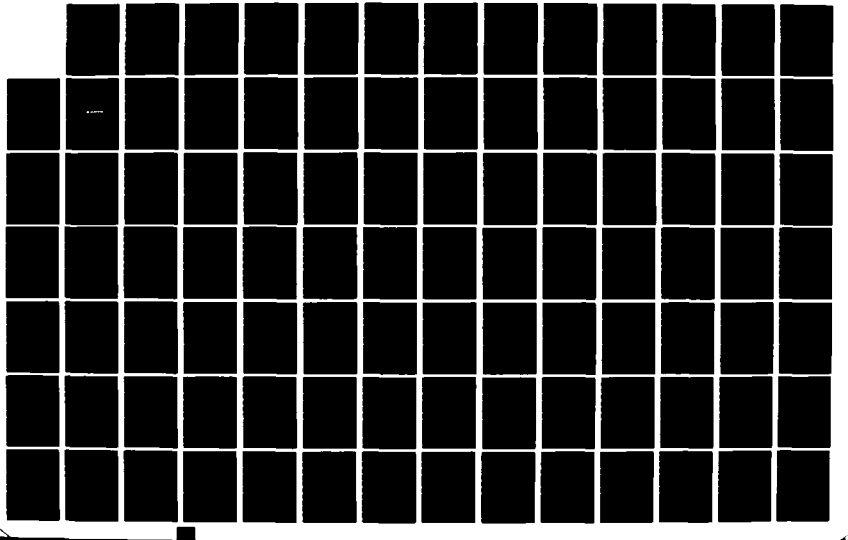
| Cluster No. | Criterion function | Increment of c. f. | Classif. % |
|---|---|---|---|
| 16 | 359 | - | 80.1 |
| 15 | 374 | 15 | 81.9 |
| 14 | 392 | 18 | 85.5 |
| 13 | 416 | 14 | 91.0 |
| 12 | 456 | 40 | 84.6 |
| 11 | 510 | 54 | 83.7 |
| 10 | 565 | 55 | 85.5 |
| 9 | 632 | 67 | 81.9 |
| 8 | 698 | 66 | 76.5 |
| 7 | 783 | 85 | 68.8 |
| 6 | 899 | 116 | 57.9 |
| 5 | 1069 | 170 | 64.3 |
| 4 | 1360 | 291 | 57.9 |
| 3 | 1756 | 396 | - |
| 2 | 2464 | 708 | - |

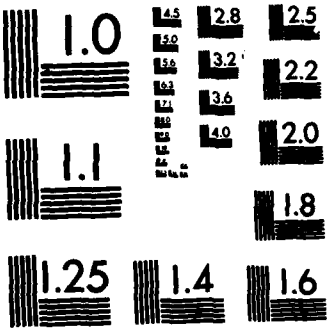
cluster number. Also shown in Table 3.1 are the recognition results for different cluster number selections. The number of clusters is equivalent to the number of primitives. The selection of 13 clusters gives the best recognition result. The $tr\ S_B$ curve which is monotonically increaseing is shown in Figure 3.5, and the $tr\ S_W$ curve which is monotonically decreasing is shown in Figure 3.6. The PFS curve is shown in Figure 3.7. The maximum PFS value appears at cluster number 13, which is identical to the selection in the previous approach.

Although there is a secondary peak at cluster number 6 in Figure 3.7, this one does not have any significant meaning. The recognition results of Table 3.1 show no indication of peak at that location. However, there does exist a secondary peak in recognition accuracy which occurs at cluster number 10. The possible reasons for these phenomena are that first, our seismic samples are not very compact and well seperated; and second, we reassign membership after each merging, this may affect the PFS value and recognition results. In spite of the secondary peak, the selection of the dominant peak gives the best results and should be the rule to follow.

The centers of the 13 clusters and the number of members in each cluster are shown in Table 3.2. The cluster centers are further plotted in the two-dimensional feature plane in Figure 3.8. Portions (17 segments) of two examples, one is a typical explosion; the other is a typical earthquake, are given in Figure 3.9, which have both original waveforms and string representations. The second segments of the two waveforms look the same but have different primitive assignment. This is because both symbol '*a*' and '*c*' have very small magnitudes compared with the other symbols (see Figure 3.8), therefore the frequency difference can

AD-A124 398 A SYNTACTIC APPROACH AND VLSI ARCHITECTURES FOR SEISMIC 2/3
SIGNAL CLASSIFICATION(U) PURDUE UNIV LAFAYETTE IN
SCHOOL OF ELECTRICAL ENGINEERING H LIU ET AL. JAN 83
UNCLASSIFIED N00014-79-C-0574 F/G 8/11 NL





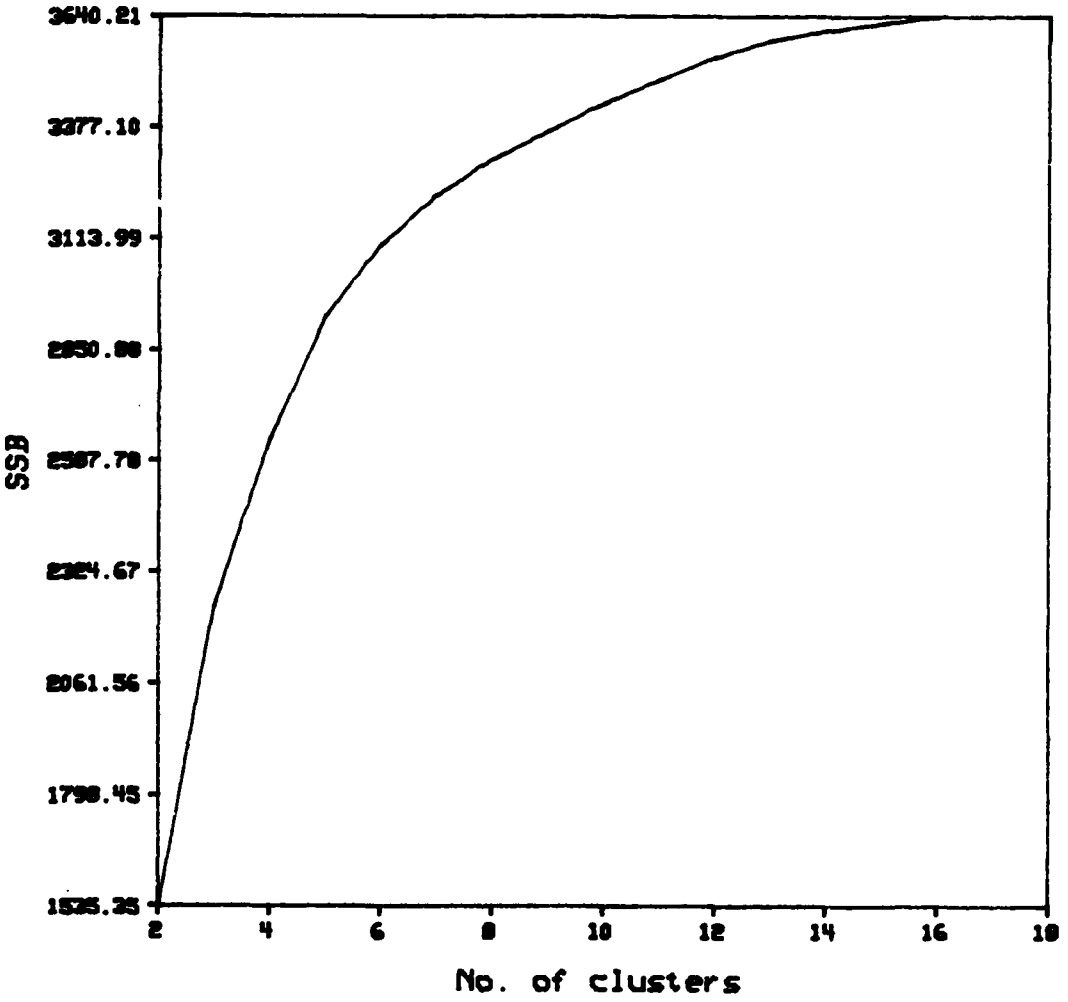



Figure 3.5 tr $S_B$ increases as the number of clusters increases.

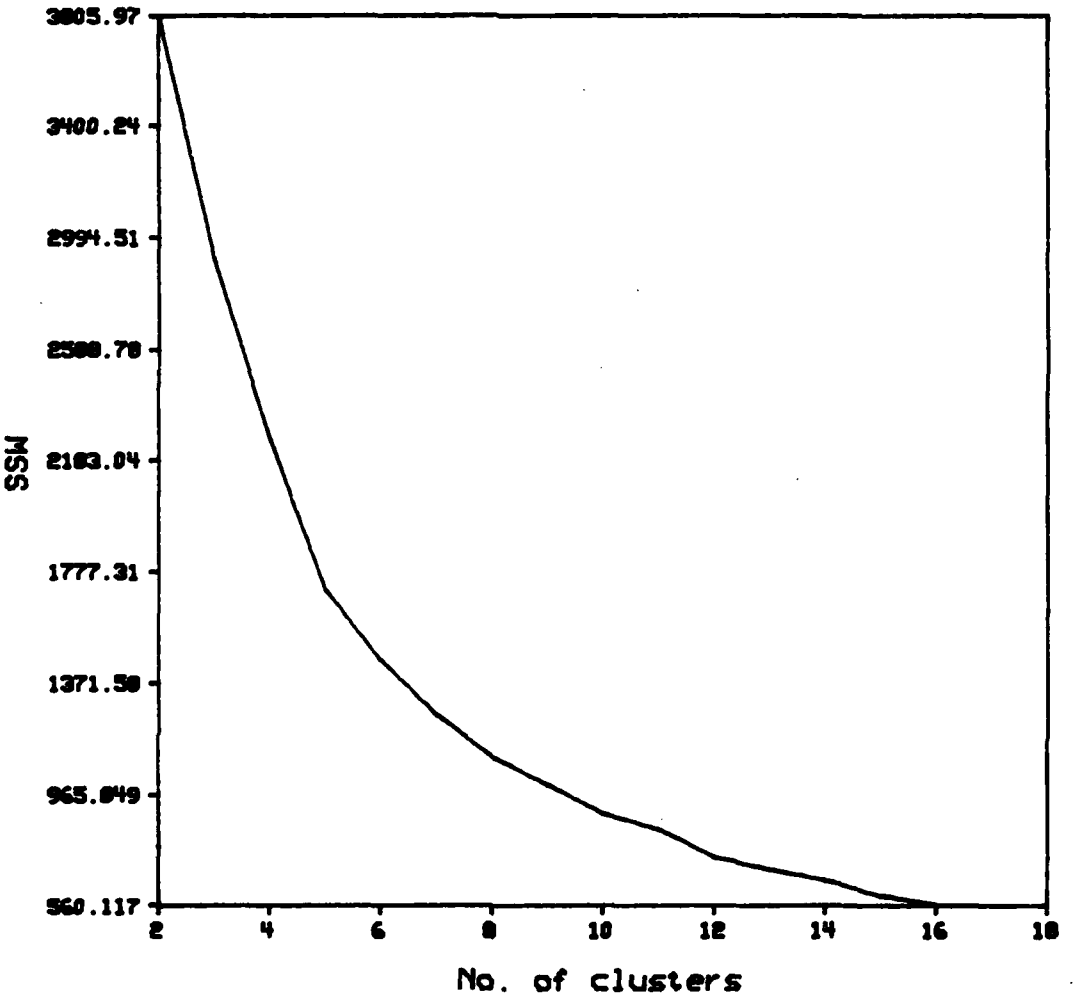



Figure 3.6 tr $S_w$ decreases as the number of clusters increases.

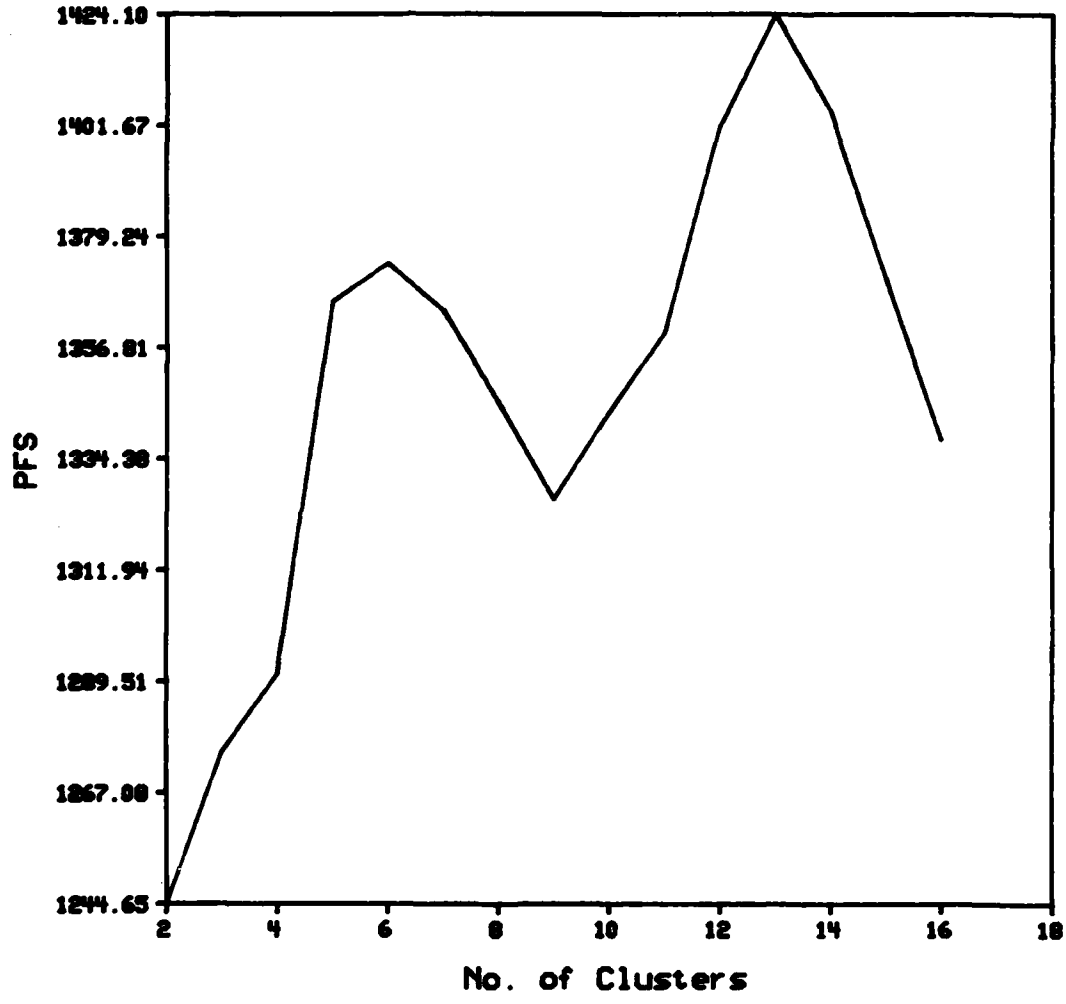



Figure 3.7 The PFS curve where the maximum value occurs at number 13.

TABLE 3.2

The center of the 13 clusters, the number of members in each cluster and the primitive symbol of each cluster.

| Cluster No. | Feature 1 (Z-C C.) | Feature 2 (L. E.) | No. of Members | Primitive Symbol |
|---|---|---|---|---|
| 1 | -1.718192 | -2.108372 | 67 | a |
| 2 | 3.336939 | -1.740116 | 36 | b |
| 3 | -.180208 | -2.387472 | 43 | c |
| 4 | -1.229273 | .987182 | 187 | d |
| 5 | .467317 | 1.048923 | 179 | e |
| 6 | .426978 | .113834 | 233 | f |
| 7 | -.407192 | 1.283638 | 209 | g |
| 8 | -.320940 | .440148 | 245 | h |
| 9 | 1.431115 | .168968 | 73 | i |
| 10 | -.306735 | -.573480 | 211 | j |
| 11 | 1.485801 | -.940290 | 145 | k |
| 12 | -1.413536 | -.255781 | 116 | l |
| 13 | .476520 | -.756842 | 256 | m |

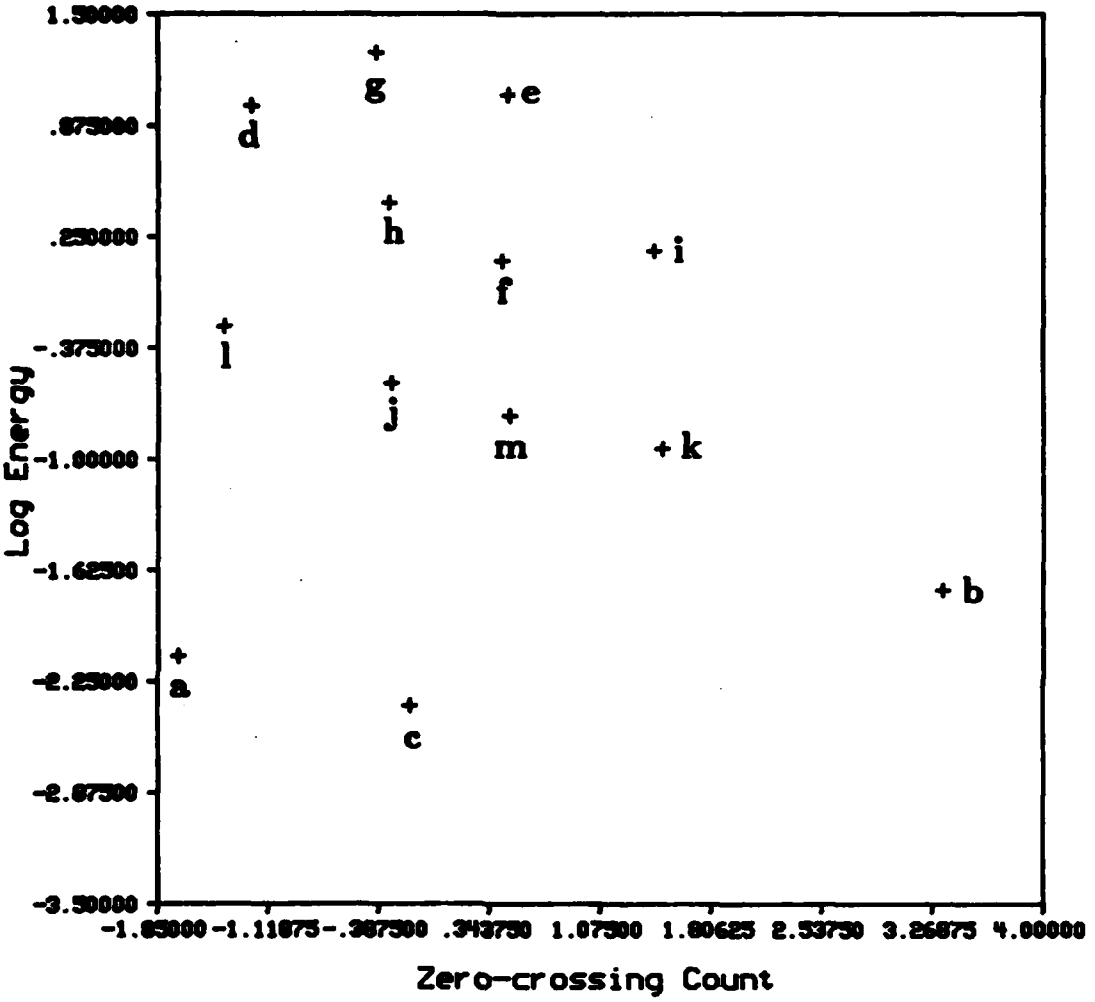



Figure 3.8 Cluster centers of the 13 clusters in the two-dimensional feature plane. The corresponding primitive symbols are also presented.

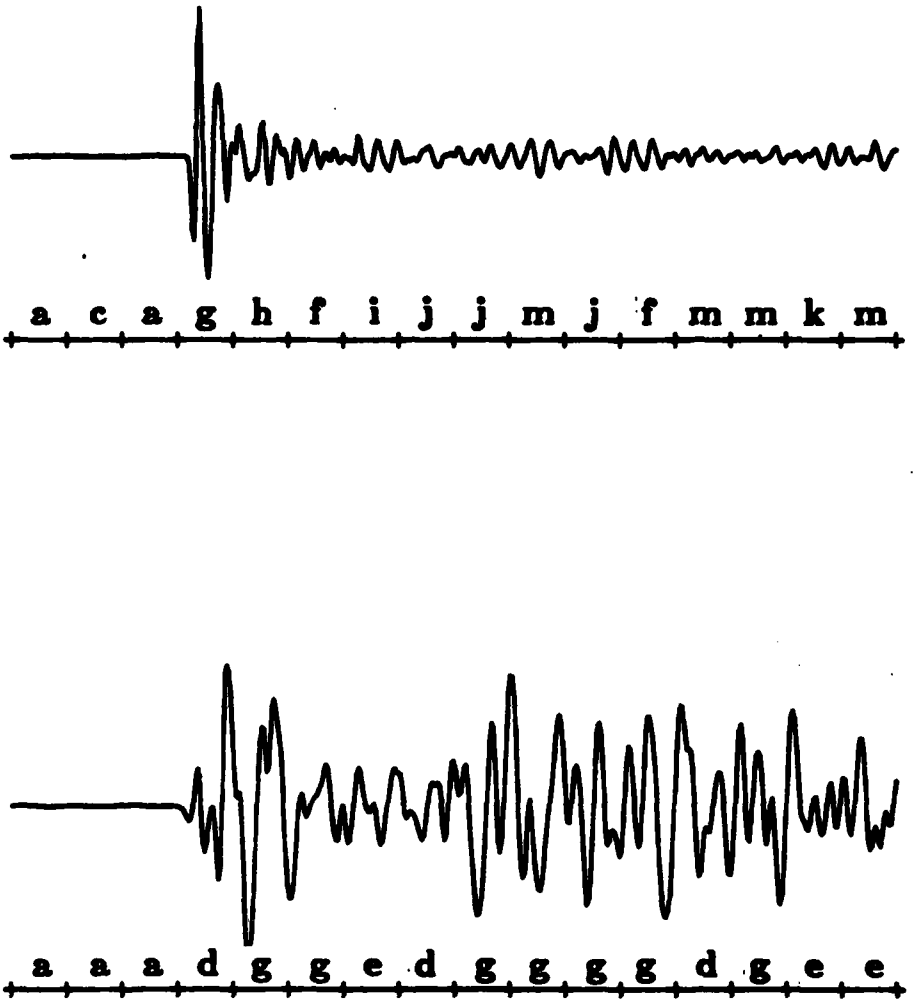



Figure 3.9 Examples of the seismic waveforms and corresponding strings. The top one is a typical explosion, and the bottom one is a typical earthquake.

not be seen due to the resolution of the drawing. Algorithm 2.1 is applied for string distance computation, and the nearest-neighbor decision rule is used for classification. Since all the records have equal length and alignment, only substitution errors are considered. The weights for substitution errors are given in Table 3.3. The weight between pattern primitives is defined as the normalized distance between corresponding clusters. Classification results and computation time of the 221 test samples are shown in Table 3.4 where 201 records are correctly classified, i.e., 91% correct rate, with an average time of 0.07 sec for each record. The experiments were run on a VAX 11/780 computer using Pascal programming language.

We use the $k$-tail finite-state inference algorithm to infer pattern grammars for the seismic waves. When $k \geq 19$, the inferred grammar is exactly the same as the canonical grammar. When $k < 19$, some equivalent states will be merged, therefore, result in fewer number of states and productions. The number of states and productions for various values of $k$ is shown in Table 3.5; it is getting smaller as $k$ gets smaller. Average parsing time of one string and percentage of correct classification for different $k$ are given in Table 3.6. The parsing time is shorter when $k$ is smaller. This is due to the smaller number of productions and states. On the other hand, the correct perentage is also smaller when $k$ is smaller. This is because derived grammars generate strings which do not belong to the positive sample set. Another reason of worse performance is that in our case only those states with longest tails are merged. In terms of transition diagram, this means only those states which are close to the initial state are merged. Because the $k$-tails of those states are empty, and only are they $k$-equivalent. This is

TABLE 3.3

**Weights for substitution error**

| | a | b | c | d | e | f | g | h | i | j | k | l | m |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 0 | 0.95 | 0.29 | 0.59 | 0.72 | 0.58 | 0.68 | 0.55 | 0.73 | 0.39 | 0.64 | 0.35 | 0.49 |
| b | 0.95 | 0 | 0.67 | 1.00 | 0.75 | 0.65 | 0.91 | 0.80 | 0.51 | 0.72 | 0.38 | 0.94 | 0.57 |
| c | 0.29 | 0.67 | 0 | 0.67 | 0.66 | 0.49 | 0.69 | 0.53 | 0.57 | 0.34 | 0.42 | 0.46 | 0.33 |
| d | 0.59 | 1.00 | 0.66 | 0 | 0.32 | 0.35 | 0.16 | 0.20 | 0.52 | 0.34 | 0.63 | 0.24 | 0.46 |
| e | 0.72 | 0.75 | 0.66 | 0.32 | 0 | 0.18 | 0.17 | 0.19 | 0.25 | 0.34 | 0.42 | 0.43 | 0.34 |
| f | 0.58 | 0.65 | 0.48 | 0.35 | 0.18 | 0 | 0.27 | 0.15 | 0.19 | 0.19 | 0.28 | 0.35 | 0.16 |
| g | 0.68 | 0.91 | 0.69 | 0.16 | 0.17 | 0.27 | 0 | 0.16 | 0.40 | 0.35 | 0.55 | 0.35 | 0.42 |
| h | 0.55 | 0.80 | 0.53 | 0.20 | 0.19 | 0.15 | 0.16 | 0 | 0.33 | 0.19 | 0.43 | 0.24 | 0.27 |
| i | 0.73 | 0.51 | 0.57 | 0.52 | 0.25 | 0.19 | 0.40 | 0.33 | 0 | 0.36 | 0.21 | 0.54 | 0.25 |
| j | 0.39 | 0.72 | 0.34 | 0.34 | 0.34 | 0.19 | 0.35 | 0.19 | 0.36 | 0 | 0.34 | 0.22 | 0.15 |
| k | 0.64 | 0.38 | 0.42 | 0.63 | 0.42 | 0.28 | 0.55 | 0.43 | 0.21 | 0.34 | 0 | 0.56 | 0.19 |
| l | 0.35 | 0.94 | 0.46 | 0.24 | 0.43 | 0.35 | 0.35 | 0.24 | 0.54 | 0.22 | 0.56 | 0 | 0.37 |
| m | 0.49 | 0.57 | 0.33 | 0.46 | 0.34 | 0.16 | 0.42 | 0.27 | 0.25 | 0.15 | 0.19 | 0.37 | 0 |

TABLE 3.4

Classification results using
nearest-neighbor decision rule

| Average time for one string (sec) | Percentage of correct classification |
|---|---|
| 0.07 | 91.0 %<br>201 records are correctly classified out of 221 |

TABLE 3.5

**The number of nonterminals, productions and negative samples accepted by the inferred grammars. The inference algorithm is k-tail algorithm with different values of k.**

| k | Explosion | | Earthquake | | No. of negative samples accepted |
|---|---|---|---|---|---|
| | Nonterm. No. | Product. No. | Nonterm. No. | Product. No. | |
| 20 | 681 | 720 | 939 | 996 | 0 |
| 19 | 681 | 720 | 939 | 996 | 0 |
| 18 | 669 | 720 | 928 | 996 | 0 |
| 17 | 641 | 692 | 900 | 970 | 0 |
| 16 | 604 | 656 | 856 | 926 | 0 |
| 15 | 566 | 618 | 804 | 874 | 0 |
| 14 | 525 | 577 | 747 | 817 | 0 |
| 13 | 484 | 536 | 688 | 758 | 0 |
| 12 | 443 | 495 | 629 | 699 | 0 |
| 11 | 402 | 454 | 570 | 640 | 0 |
| 9 | 320 | 372 | 452 | 522 | 0 |
| 7 | 238 | 290 | 334 | 404 | 0 |
| 5 | 156 | 208 | 216 | 286 | 0 |

TABLE 3.6

The average parsing time and percentage of correct classification of the error-correcting parsers with different values of k.

| k | Average parsng time for one string (sec) | Percentage of correct classification (%) |
|---|---|---|
| 20 | 2.55 | 91.0 |
| 19 | 2.55 | 91.0 |
| 18 | 2.72 | 84.2 |
| 17 | 2.67 | 81.0 |
| 16 | 2.54 | 73.8 |
| 15 | 2.33 | 72.8 |
| 14 | 2.15 | 71.0 |
| 13 | 2.10 | 69.7 |
| 12 | 2.03 | 69.7 |
| 11 | 1.83 | 71.0 |
| 9 | 1.47 | 69.2 |
| 7 | 1.15 | 68.8 |
| 5 | 0.77 | - |

the consequence when all the training samples have equal length. Normally, the merged states should distribute uniformly between initial and final states. One final note about Table 3.6 is that the decrease of parsing time is true for any cases, but the decrease of correct percentage may not be true for other cases because the experimental results of our limited data set are neither representative nor conclusive.

We also try the tail-clustering finite-state inferene algorithm. Since there are no two states which have common sentences, therefore no merge occurs. The productions and nonterminals are the same as those of $k$-tail algorithm with $k$ = 20. Again, this is due to the characteristics of this specific data set, and should not be interpreted against the algorithm itself. We can modify the condition for merge so that two states are merged when the distance between some of their member sentences is less than a threshold. This will guarantee a reduction of grammar size, but again the recognition results may be unpredictable.

### 3.6 An Application of Syntactic Seismic Recognition to Damage Assesment

Damage assesment of a structure after strong earthquake is a very complex problem (Yao, 1979). It is usually performed by a structural engineering expert who makes his or her judgement by personal experience and professional knowledge. The key informations include characteristics of the structure, observable damages, seismic (vibration) recordings and nondestructive testing results. Ishizuka et al. (1981) have proposed a rule-based damage assesment system which employs the fuzzy set theory and the production system with certainty factor to

infer the damage state. Its performance relies on proper assignment of membership function and design of inference rules. The pattern recognition techniques can also be applied to damage assesment, which is based on the analysis of seismic recordings. Its advantages are easy to implement and contains no uncertainty factor.

Seismic recordings, i.e., acceleration and/or displacement recordings, are the only records which show the detailed response of the structure during a strong earthquake. It is quantitative, complete and objective. Therefore, if we want to apply pattern recognition techniques to damage assesment, the seismic recordings are very good candidates. A structure without damage will behave stiffer than the one with damage. Therefore from the seismic recording, preferably displacement recording for the reason of no high frequency noise, we can tell the relative degree of damage.

Since each building is different in structure, we have to make assesment individually. One possible solution is to compare the top level displacement with the basement displacement. The basement displacement represents the ground motion, i.e., the input to the building. The deformation distance between these two waveforms will be small if the building is damaged; otherwise, the deformation distance will be large. Unfortunately good training samples are unavailable so far. The real recordings are not only insufficient but also unclassified. However, there are a few experimental data from the laboratory which can be used as a starting point.

Figure 3.10 shows the top level displacement and basement acceleration (at the bottom) during a simulated earthquake test on the model of a ten-story reinforced concrete building. There are totally

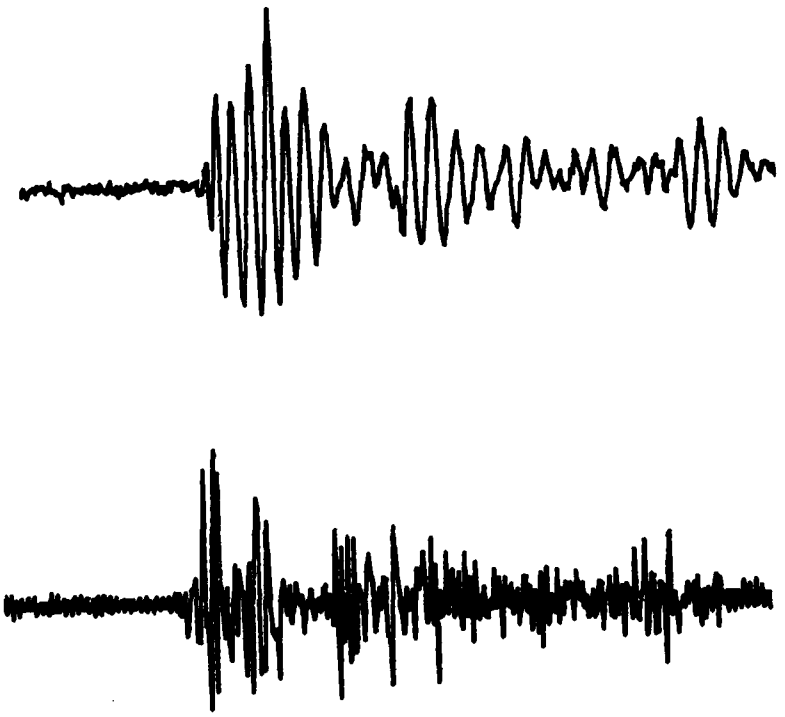



Figure 3.10 Top level displacement and basement acceleration (bottom).

seven test runs. It is obvious from Figure 3.10 that the accleration waveform is much more complicated than the displacement waveform. Since they are convertible, we chose displacement seismogram for comparison.

Since only the basement acclerations are available, we have to compute displacements using numerical integration. The basement displacements of the seven runs are all the same as shown in Figure 3.11, only the magnitudes are intensified from run to run so as to assure more damage after more runs. The top level displacements are shown in Figure 3.12. It is not diffucult to see that the top level displacement of run seven is more similar in figuration to basement displacement than the top level displacement of run one is. This shows that the building structure becomes softer due to the cracks, breaks and other implicit damages. Some potential damages may not be seen from the appearence of the building, but they will be shown on the seismic recording since it reflects the actual structure response. This is one of the reasons why the analysis of seismic recording is important. The other reason is that we can compute the similarity, or deformation distance on the other hand, between the waveforms which can be further used in a knowledge-based damage assesment system.

Computation of the deformation distance between the seismic waveforms are based on the modified dynamic time warping distance in Section 2.2. Comparing Figure 2.6 with Figure 3.12 we will find that the waveforms in Figure 2.6 are actually taken from those in Figure 3.12. The slope constraints and local distance functions are shown in Figure 3.13.

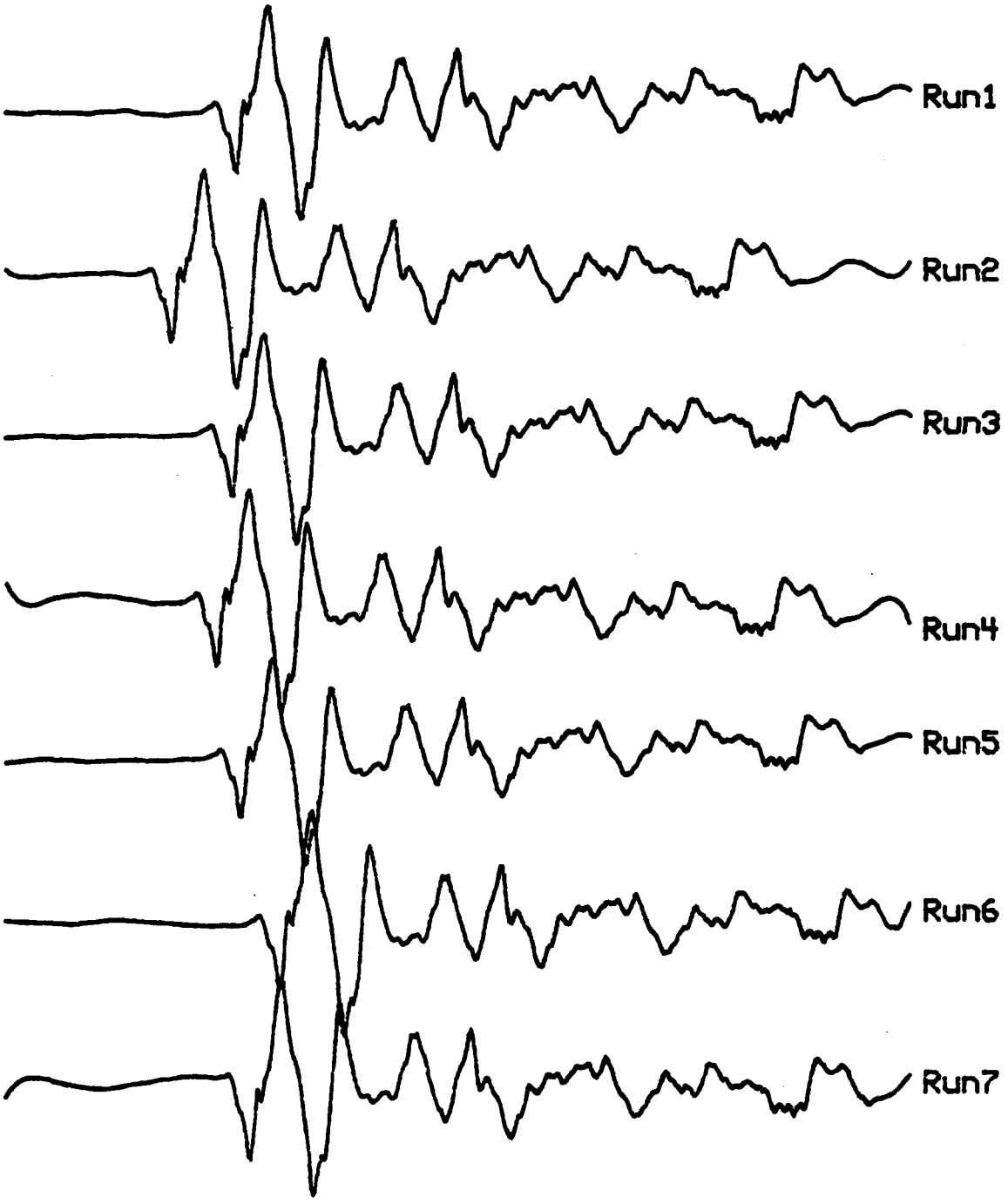



Figure 3.11 Basement displacement of the seven test runs.

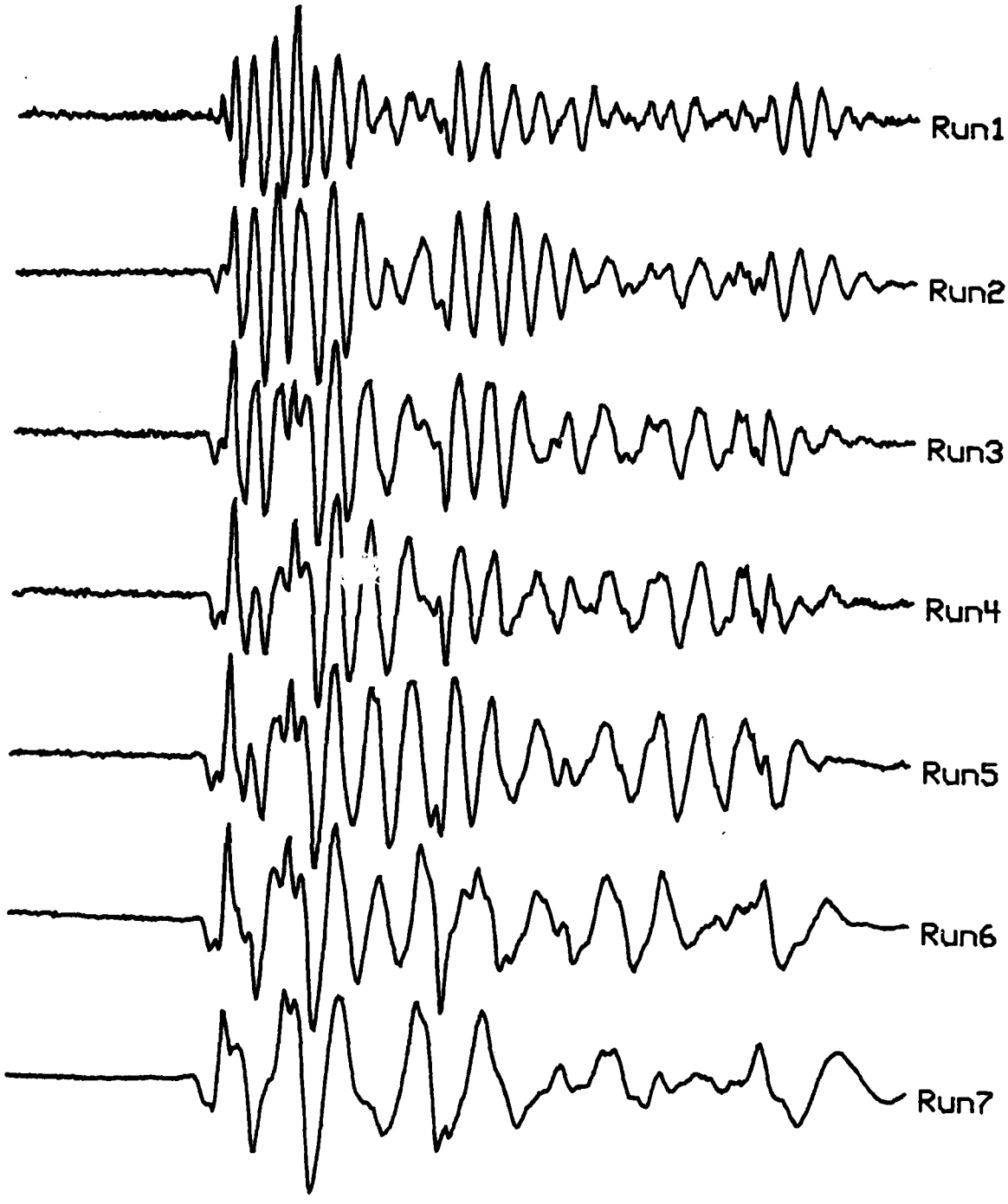
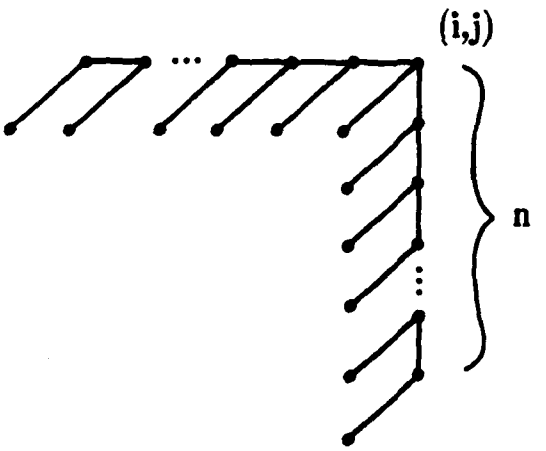



Figure 3.12 Top level displacement of the seven test runs.




$$\delta[i,j] = \min \begin{bmatrix} \delta[i-1,j-l] + |a_i - \sum_{k=1}^{l} b_{j-k-1}| \\ l = 1,2,\ldots,n \\ \delta[i-l,j-1] + |\sum_{k=1}^{l} a_{i-k-1} - b_j| \\ n \geq 2 \end{bmatrix}$$

Figure 3.13 Diagram of slope constraints and local distance function for string distance computation in damage assesment application.

The selection of string representation and the selection of computational algorithm for string deformation distance are correlated. We observed from the waveforms in Figure 3.12 that several local peaks are deformed and merge into a large peak. Therefore, we consider each peak as a component, i.e., primitive or symbol, of string representation. The next problem is how to describe each peak. Of course, shape and geometric properties can describe a peak, they are far complicated than what is needed. Besides, it is difficult to implement these features in distance computation. The area of each peak contains the informations about the duration and amplitude of the peak. Since different combinations of duration and amplitude may have same area, area alone is ambiguous. But we don't need to worry about this problem since we are dealing with recordings from the same structure, such randomly contrast shapes will not occur. We developed a special string deformation distance computation for this application, which is a modified dynamic time warping distance as shown in Section 2.2.1. The type of this deformation distance is ordinal, i.e., rank orders have meaning, and interval, i.e., seperation between numbers is meaningful. However, the lower and upper bounds of this distance is open, i.e., the distance is in the interval $(0, M)$ where $M$ is the summation of the total area of the two strings. For example, if $x = a_1 a_2 ... a_m$, and $y = b_1 b_2 ... b_n$ then

$$M = \sum_{i=1}^{m} a_i + \sum_{j=1}^{n} b_j.$$

Each seismic waveform $x$ is converted into a string of real numbers, $x = a_1 a_2 ... a_n$, $a_i > 0$, such that the $i$th component of the string, $a_i$, represents the area of the $i$th peak. The definition of the

*peak here is the segment between two adjacent zero-crossing points.* Therefore one peak may contain many local maxima and minima. It often happens that small ripples and zero-crossings may exist due to the noise. These noisy ripples can be removed by setting a threshold $T$. Only those peaks whose areas are larger than threshold $T$ are considered as effective components. The waveforms are scanned from both side until a peak larger than $T$ is reached on each direction. The leftmost peak larger than $T$ will be the first component of the string and the rightmost peak larger than $T$ will be the last component of the string. This process will eliminate the noisy ripple before and after the signal. The noisy ripples within the signal are combined with the nearest peak which is greater than $T$. Therefore, only the significat peaks are converted into components of the string. The algorithm for computing string deformation distance is similar to that of Sakoe and Chiba's, only the slope constraints and local distance functions are different.

The deformation distance between the basement displacement and the top level displacement of each run is plotted in Figure 3.14. Since each run of the test adds some damage to the structure, the degree of damage is proportional to the number of tests. Greater damage makes the structure softer, consequently the deformation distance between the basement waveform and top level waveform is smaller. It can be seen from Figure 3.14 that the deformation distance is getting smaller after more runs of tests. Figure 3.14 also shows that large damage occurs during the first three runs since the differences of the deformation distance, i.e., the slope, are larger than those of the later runs.

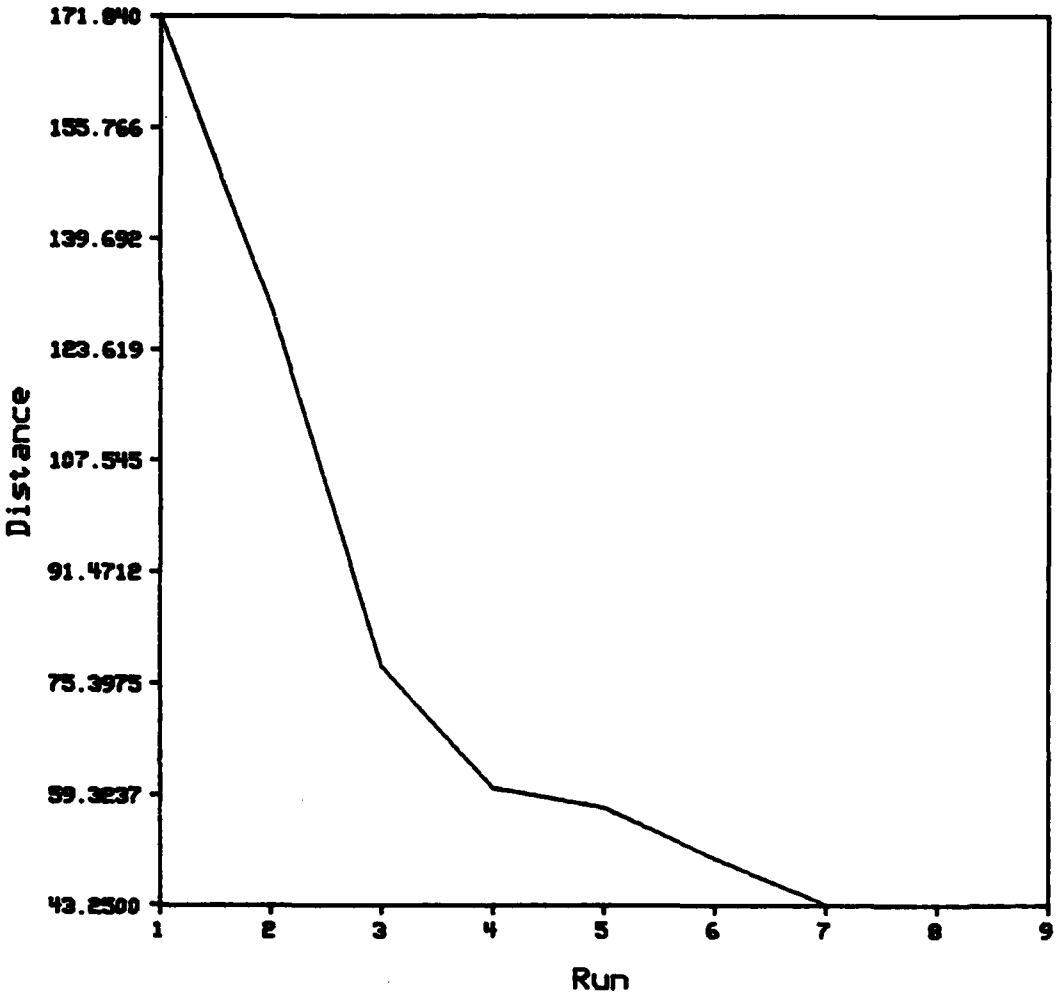



Figure 3.14 Distance between the basement displacement waveform and the top level displacement waveform of each run.

In order to normalize the length of the strings come from different event, the deformation distance in Figure 3.14 can be divided by the length of the basement waveform so that the deformation distance of different event can be compared. The domain of damage can be divided into several intervals, for example, negligible, slight, moderate, severe, etc. The deformation distance is used for classification of damage degree. The classification depends on which category the deformation distance of one event falls into. Other informations such as human observations and system identification results are usuful auxilary informations, for example, to resolve the conflict when the distance falls at the boundary. But system identification is a very complicated matter, it is mainly for the study of system characteristics. Visual informations are easy to obtain and are helpful in resolving conflict and ambiguity.

The proposed system does not have the opportunity to test real data because of the lack of data. The research in damage assesment is only in its infancy. No organization or individual has been working on the collection and classification of the real data. We must understand that appropriate samples for damage assesment are rather difficult to obtain. The structure must be equipped with recording devices, subject to strong earthquake and bear certain degree of damage. Therefore, the demonstration of the proposed method is based on experimental data only. It attempts to show the feasibility instead of practicability of the proposed method.

The segmentation of waveform employs some structural (contextual) information. Peak extraction needs structural information, merge of small peaks with the nearest large peak also needs structural information. In our demonstration, only the top level recordings are

used for comparison. Intermediate levels are similar to top level but with smallar amplitude.

## 3.7 Conclusion

In this chapter, syntactic pattern recognition has been applied to the discrimination of earthquake and nuclear explosion based on seismic waveforms. The waveforms are segmented by a fixed length. A clustering procedure classifies these segments and a symbol is assigned to each cluster. Finite-state grammars are inferred from the training set using $k$-tail inference algorithm. An error-correcting parser and a nearest-neighbor rule are compared with respect to their performance in recognition speed and accuracy. Although the classification results seem to be encouraging, there is plenty of room for improvement. The selection of a set of distinguishing features is the most important part in practical pattern recognition applications. The difficulty increases when the classes are somewhat overlapped. Most of the features which are effective in decision-theoretic approach can also be used in the syntactic approach for primitive recognition. The number of features selected should be kept as small as possible for the sake of computational efficiency.

In string distance computation, the assignment of weights for transformation errors is a difficult subject especially when insertion, deletion and substitution are all included. The seperation between clusters can be used as the substitution weights between corresponding primitives as we did in our experiment. The distance from a cluster center to the origin can be used as the insertion and/or deletion weight

of that primitive. Heuristic information may be necessary and helpful in most cases.

Syntactic approach can be modified to deal with stochastic models if the probabilities associated with pattern classes and training samples can be easily determined. In this case, there will be stochastic grammar, stochastic language and maximum-likelihood parsing (see Fu, 1982). We did not apply the stochastic approach because the class and string probabilities are unavailable. This must be done from the analysis of the previous records. If the probabilities can be determined precisely, which can be made to a certain degree, the class-overlap problem can be solved. Syntactic approach can be made more flexible by adding numerical information (attribute) to the primitives. Meanwhile, it can also make the pattern grammar less complex. We will discuss an attributed seismic grammar and its parsing in Chapter IV.

At the present stage, our experiments show that the nearest-neighbor decision rule is faster than the error-correcting parsing. Although the speed of error-correcting parsing depends on the structure of the grammar, the nearest-neighbor rule is faster in general. VLSI architectures have been recently applied to both string matching and recognition (by parsing), which will be discussed in Chapter V. Decision between simple, faster classification and sophisticated, slower syntax analysis should be made according to application requirements.

Syntactic pattern recognition has also been applied to damage assesment where the seismic recordings are the physical measurements. Strings of various length are constructed from the seismic waveforms. A modified dynamic time warping is developed for computing the string distance. The segmentation of waveform in syntactic

pattern recognition usually uses shape information. The shape information appears to be not important for seismic signal. Besides, it does not have much discrimination capability. The envelops of the signal appears to be very good features in some cases, for example, considering Figure 1.2, but not so in other cases, for example, when Figure 1.3 and 1.4 are compared with Figure 1.2. The application to damage assesment shows that special algorithm for string distance computation must be developed for some applications when the general string distances seem unable to solve the problem.

# CHAPTER IV

# INFERENCE AND PARSING OF ATTRIBUTED GRAMMAR FOR SEISMIC SIGNAL RECOGNITION

## 4.1 Introduction

Attributed grammars were first formulated by Knuth (1968) where "meaning" can be assigned to a string in a context-free language by defining "attributes" of the symbols in a derivation tree for that string. The attributes are defined by functions associated with each production in the grammar. Although the idea of attributed grammar is due to Irons (see Knuth, 1968), Knuth included inherited attributes as well as synthesized attributes which often leads to significant simplification. While attributed grammars were originally proposed for programming languages, they have been applied to pattern recognition recently and increasingly. Tang and Huang (1979) used attributed grammars for image understanding. You and Fu (1978, 1979), Tsai and Fu (1980) and Tai and Fu (1981) have applied attributed grammars to shape recognition and transformation. Shi and Fu (1982) proposed an efficient error-correcting parser for attributed tree grammars where semantic information are associated with each terminal but no semantic rule is associated with the production. Leung (1982) also proposed an error-correcting parser for attributed grammars with applications to character recognition. Knuth's formal semantics can also be applied to

patterns described by picture description language (PDL) expressions (Fu, 1982).

The advantages of using attributed grammars for pattern recognition are twofold. The inclusion of semantic information increases the flexibility in pattern description; in the meantime, it reduces the syntactic complexity of the pattern grammar. We may notice that all the above applications are essentially to pictorial shape recognition where length and angle are useful semantic informations. This same set of attributes can also be used in waveform shape recognition, e.g., ECG analysis, where shape information is very important in recognition. However, they can not be applied to the signals, e.g., EEG, seismic and speech, where shape informations are not particularly important. The segmentation of these signals usually corresponds to a short, fixed- or variable-length time period. In order not to overlook any transition, the time periods are usually kept relatively short. Therefore, it is very common that the same primitive may last for several periods. This often makes the pattern strings and the inferred grammars unnecessarily complicated. The numbers of productions and nonterminal symbols are usually very large as we can see from the experimental results in Section 3.5. Instead of keeping track of all these identical primitives, we can use one syntactic symbol to represent the type of the primitive with an attribute to indicate the length of the primitive. This leads to the application of length attribute to seismic and other similar digital signal analysis.

A pattern primitive $a$ can be represented by a 2-tuple

$$a = (s, x)$$

where $s$ is a syntactic symbol denoting the primitive structure of $a$, and $x = (x_1, x_2, \ldots, x_m)$, $m \geq 0$, is an $m$-dimensional semantic vector with each $x_i$, $i = 1, 2, \ldots, m$, denoting a numerical measurement. A pattern string can be represented by $a_1 a_2 a_3 \ldots a_k$, where $a_i = (s_i, l_i)$, $l_i$ is the length of primitive $a_i$, $1 \leq i \leq k$. For a fixed-length segmentation, $l_i = c$ for all $i$, where $c$ is a constant. For a variable-length segmentation, $l_i$ may or may not equal to $l_j$ when $i \neq j$. In our case, $l_i = c$ for $1 \leq i \leq 20$, where $c = 60$ points. For simiplicity, with constant length in mind, we can eliminate the semantic part. For example, a pattern string may look like

$a\ a\ a\ d\ g\ g\ g\ e\ g\ g\ g\ g\ g\ g\ g\ e\ g\ e\ e\ g$

where these are syntactic symbols. It can be further simplified by merging identical symbols, therefore the above string becomes

$a\ d\ g\ e\ g\ e\ g\ e\ g$

3 1 3 1 7 1 1 2 1

where the numbers are numbers of unit lengths; each unit length contains 60 points in our case. This idea shows some storage improvement in string representation, and it will show significant improvement in grammatical inference as we will see in the next section. Although we used finite-state grammars to describe the seismic patterns in Chapter III, we will use attributed cfg's here. This is because attributed fsg's do not have much reduction in the number of productions and nonterminals. Only attributed cfg's can drastically reduce the production number, therefore make the recognition more efficient. An error-correcting parser for attributed context-free grammar is given in Section 4.3. Stochastic attributed grammar and parsing will be discussed in Section 4.4.

## 4.2 Inference of Attributed Grammar for Seismic Signal Recognition

An attributed context-free grammar is a 4-tuple G = ($V_N$, $V_T$, P, S) where each production rule contains two parts, one is a syntactic rule, the other is a semantic rule (Knuth, 1968). Each symbol $X \in (V_N \cup V_T)$ is associated with a finite set of attributes $A(X)$; and $A(X)$ is partitioned into two disjoint sets, the synthesized attribute set $A_0$(X) and the inherited attributed set $A_1$(X). The syntactic rule has the following form

$$X_{k0} \rightarrow X_{k1}X_{k2}\ldots X_{kn_k}$$

where k means the $k$th production. The sematic rule maps values of certain attributes of $X_{k0}$, $X_{k1}$, ..., $X_{kn_k}$ into the value of some attribute of $X_{kj}$. The evaluation of synthesized attributes is based on the attributes of the descendents of the nonterminal symbol, therefore it is a bottom-up fashion in the tree structure. On the contrary, the evaluation of inherited attributed is based on the attributes of the ancestors, therefore it is a top-down fashion in the tree structure.

In Chapter III, we have chosen a set of 41 explosion seismic records as training samples. Each record has been converted into a string of 20 primitives. If we use the k-tail algorithm to infer a finite-state grammar for the pattern class with a value of $k$ = 20, the total number of productions will be 720 and the number of nonterminal symbols will be 681. In order to reduce the size of the grammar we use one length attribute, i.e., the number of unit lengths. The input strings are attributed strings, and the production rule of the grammar has a syntactic part as well as a semantic part which contains both synthesized and inherited attributes. The type of grammar is also upgraded into a

context-free grammar, due to the type of $S$-productions. Tai and Fu (1982) used the length attribute of the strings in the inference of a class of context-free programmed grammar (cfpg). However, the length attribute is only for the construction of the control diagram, i.e., a graphical representation of the success and failure go-to fields. The inferred cfpg is nonattributed, and the parsing was not discussed. We use length attribute in both inference and parsing. The inferred grammars are attributed grammars, and the attribute plays an important role in parsing.

To explain our inference procedure, let us first consider one input string

*a a a d g g g e g g g g g g g e g e e g*

where each primitive has a length attribute 1 which means 1 unit length. First, it will be converted into the following string by merging identical primitives.

*a d g e g e g e g*

3 1 3 1 7 1 1 2 1

Theoretically, the length attribute is continuous. But in digital signal processing, the waveforms represenesented by a finite number of sampled points, therefore, the length is always discrete in practical cases. In our case, the length attribute is the number of unit lengths. It is discrete and is a positive integer. Then we can infer the following attributed grammar

| | Syntactic rules | Semantic rules |
|---|---|---|
| (1) | $S \rightarrow ADGEGEGEG$ | $L(A1)=3, L(D)=1, L(G1)=3,$ |
| | | $L(E1)=1, L(G2)=7, L(E2)=1,$ |

$L(G3)=1, L(E3)=2, L(G4)=1$

| | | |
|---|---|---|
| (2) | $A \rightarrow aA$ | $l(A1)=l(a)+l(A2)$ |
| (3) | $A \rightarrow a$ | $l(A)=l(a)$ |
| (4) | $D \rightarrow dD$ | $l(D1)=l(d)+l(D2)$ |
| (5) | $D \rightarrow d$ | $l(D)=l(d)$ |
| (6) | $E \rightarrow eE$ | $l(E1)=l(e)+l(E2)$ |
| (7) | $E \rightarrow e$ | $l(E)=l(e)$ |
| (8) | $G \rightarrow gG$ | $l(G1)=l(g)+l(G2)$ |
| (9) | $G \rightarrow g$ | $l(G)=l(g)$ |

where L denotes inherited length attribute, $l$ denotes synthesized length attribute and the number right after the nonterminal symbol is used to distinguish between occurrences of like nonterminals. It is noted that the inherited attributed L does not pass down to the descendents as it usually does; rather it is used to maintain the semantic information of the training string and as a reference for comparison in parsing. For simplicity we let $l(a) = 1$ for all $a \in V_t$. When we have another input string

$a\ a\ c\ d\ e\ h\ i\ h\ f\ f\ f\ f\ f\ f\ h\ m\ f\ f\ f\ f$

we convert it into

$a\ c\ d\ e\ h\ i\ h\ f\ h\ m\ f$

2 1 1 1 1 1 1 6 1 1 4

and add to the grammar the following productions

| Syntactic rules | Semantic rules |
|---|---|
| $S \rightarrow ACDEHIHFHMF$ | $L(A)=2, L(C)=1, L(D)=1,$ $L(E)=1, L(H1)=1, L(I)=1,$ $L(H2)=1, L(F1)=6, L(H3)=1,$ $L(M)=1, L(F2)=4$ |
| $C \rightarrow cC$ | $l(C1)=l(c)+l(C2)$ |
| $C \rightarrow c$ | $l(C)=l(c)$ |
| $H \rightarrow hH$ | $l(H1)=l(h)+l(H2)$ |
| $H \rightarrow h$ | $l(H)=l(h)$ |
| $I \rightarrow iI$ | $l(I1)=l(i)+l(I2)$ |
| $I \rightarrow i$ | $l(I)=l(i)$ |
| $F \rightarrow fF$ | $l(F1)=l(f)+l(F2)$ |
| $F \rightarrow f$ | $l(F)=l(f)$ |

We may notice that after reading a few input strings there will be no need to add those $C \rightarrow cC$, $C \rightarrow c$ productions. We only need to add one production for each input string, i.e., the first production in the above example. In fact, there are $2m+n$ productions for a set of $n$ training strings, where $m$ is the number of nonterminal symbols. We now formulate the inference algorithm of attributed grammars which use length attribute.

*Algorithm 4.1* Inference of Attributed Seismic Grammar Using A Length Attribute

*Input:* A set of training strings where each string has a syntactic symbol and a length attribute.

*Output:* An Attributed Grammar.

*Method:*

(1) For each input string, merge identical primitives; the length is the summation of the individual lengths.

(2) For each input string $a_1a_2a_3...a_k$, add to the grammar the production $S \rightarrow A_1A_2A_3...A_k$ where $A_i$ is the nonterminal corresponding to terminal $a_i$; and the semantic rule $L(A_i) = l_i$, $1 \leq i \leq k$, where $l_i$ is the length attribute of primitive $a_i$.

(3) For each primitive $a$, add to the grammar the production $A \rightarrow aA$, $l(A_1) = l(a) + l(A_2)$ and $A \rightarrow a$, $l(A) = l(a)$, if they are not already existed.

(4) The set of terminals includes all the different primitives; the set of nonterminal includes all the nonterminal symbols in Step (2).

A flow chart of this inference algorithm is given in Figure 4.1. This inferred grammar will generate excessive strings if we apply syntactic rules only. However, we can use semantic rules (inherited attributes) to restrict the grammar so that no excessive strings are generated.

The inferred grammar from the 41 training strings is shown in the following.

| | Syntactic rules | Semantic rules |
|---|---|---|
| ( 1) | $S \rightarrow ACAGHFIJMJFMKMJM$ | (1,1,1,1,1,1,1,2,1,1,1,2,1,1,3,1) |
| ( 2) | $S \rightarrow MKLGIFDIFHFMKILIB$ | (1,1,1,1,1,1,1,1,1,1,2,1,1,1,2,1,2) |
| ( 3) | $S \rightarrow LEIFJLFBFHDJFKJL$ | (3,2,1,1,1,1,1,1,1,1,1,2,1,1,1,1) |
| ( 4) | $S \rightarrow LJLEFKJHFJMJMIFJ$ | (1,1,1,1,1,2,1,1,3,1,1,1,1,1,1,2) |
| ( 5) | $S \rightarrow LJLGFHFHFHIFMJFLFM$ | (1,1,1,1,1,1,1,2,1,1,1,2,1,1,1,1,1,1) |
| ( 6) | $S \rightarrow ACDEHIHFHMF$ | (2,1,1,1,1,1,1,6,1,1,4) |
| ( 7) | $S \rightarrow ALGIMLMKJMLMJLJL$ | (1,2,1,2,1,1,1,1,1,1,3,1,1,1,1,1) |
| ( 8) | $S \rightarrow LMLGEMKJKMKJKMJM$ | (1,1,1,1,1,1,2,1,1,1,1,1,1,1,3,2) |

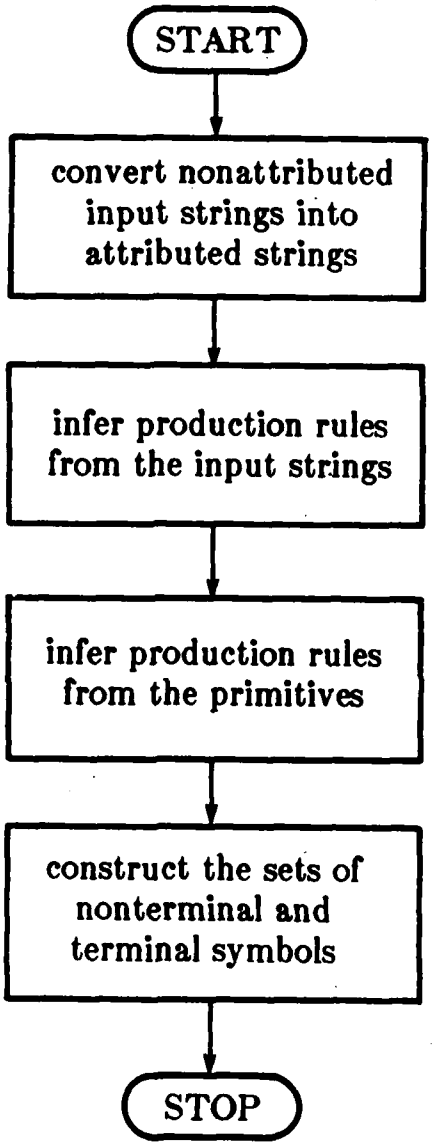



Figure 4.1 A flow chart of the inference algorithm (Algorithm 4.1).

| | | |
|---|---|---|
| ( 9) | $S \rightarrow CKDIFKJMKMJMJM$ | (1,2,1,1,4,1,1,1,1,1,2,2,1,1) |
| (10) | $S \rightarrow DLDHJMLMJFLMKL$ | (2,1,1,2,1,3,1,1,1,1,1,1,2,2) |
| (11) | $S \rightarrow CACEIFKMKJMKM$ | (1,1,1,1,1,1,1,1,3,2,5,1,1) |
| (12) | $S \rightarrow LMGFKFIFMJM$ | (2,1,1,1,1,1,2,3,3,4,1) |
| (13) | $S \rightarrow ABCGIMKMKBKJM$ | (1,1,1,1,1,4,1,2,2,2,1,1,2) |
| (14) | $S \rightarrow CEHIJFMFKJMFMFJ$ | (3,1,1,1,1,1,2,2,1,1,2,1,1,1,1) |
| (15) | $S \rightarrow KMKEFMFIJKMJKJKM$ | (1,1,1,1,2,1,1,1,1,1,1,1,3,1,1,2) |
| (16) | $S \rightarrow LJEHDFLJMFJLJFJ$ | (2,1,2,1,1,1,2,2,1,1,1,1,1,2,1) |
| (17) | $S \rightarrow JMGFHMFHFMHLJIM$ | (1,2,2,2,1,1,1,2,1,2,1,1,1,1,1) |
| (18) | $S \rightarrow BJEFKMKMKMKMKMK$ | (2,1,2,1,1,1,2,1,1,3,1,1,1,1,1) |
| (19) | $S \rightarrow BCBGHEFHFJF$ | (1,1,1,2,3,1,1,7,1,1,1) |
| (20) | $S \rightarrow IKEIHIFIHFIHFLF$ | (1,2,1,1,1,1,2,1,1,4,1,1,1,1,1) |
| (21) | $S \rightarrow DFHFDFLIF$ | (6,2,1,1,1,2,3,2,2) |
| (22) | $S \rightarrow ACEHFJMKFJMKM$ | (1,2,1,2,3,1,3,1,1,1,2,1,1) |
| (23) | $S \rightarrow JLGHDHLMJL$ | (1,2,1,1,1,1,7,1,4,1) |
| (24) | $S \rightarrow KMBEHFMKBKM$ | (1,1,1,2,1,1,1,4,2,5,1) |
| (25) | $S \rightarrow KBKGHMFMFKHMJ$ | (1,1,1,1,1,1,2,2,2,1,1,5,1) |
| (26) | $S \rightarrow LMIEIHFHJIKMLKLK$ | (1,2,1,1,1,1,1,1,1,1,1,1,3,1,2,1) |
| (27) | $S \rightarrow ADGEGEGEG$ | (3,1,3,1,7,1,1,2,1) |
| (28) | $S \rightarrow MACGHFJMFJMJM$ | (1,1,1,1,1,1,1,3,1,1,2,2,4) |
| (29) | $S \rightarrow JMGEFKJMKJKMJ$ | (2,1,1,1,1,2,2,5,1,1,1,1,1) |
| (30) | $S \rightarrow LDGEDHDLDLDLD$ | (1,2,1,1,1,1,2,1,1,1,1,2,5) |
| (31) | $S \rightarrow IHFEIEHIHIFIFIDI$ | (1,1,1,2,1,1,1,1,1,2,1,2,1,2,1,1) |
| (32) | $S \rightarrow HIEIEIFHDHDEBFHF$ | (1,2,2,1,1,2,1,1,1,1,1,1,1,2,1,1) |
| (33) | $S \rightarrow GDEGEIEGIGEDGDE$ | (1,1,1,1,2,1,5,1,1,1,1,1,1,1,1,1) |
| (34) | $S \rightarrow KBHDGHDHGDGDGHD$ | (2,1,1,1,1,1,1,1,1,4,1,1,2,1,1,1) |
| (35) | $S \rightarrow ACAGEFIFKFMFMJM$ | (1,1,1,1,1,3,1,2,1,1,3,1,1,1,1) |

| | | |
|---|---|---|
| (36) | $S \rightarrow LJLGIFLFMJFLFMJMJF$ | (1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1) |
| (37) | $S \rightarrow DFDHDHDLDF$ | (4,1,2,1,2,4,1,1,3,1) |
| (38) | $S \rightarrow HJLFEFGEGFIFEH$ | (1,1,1,1,1,1,1,5,2,1,1,1,2,1) |
| (39) | $S \rightarrow FJILFHGEIEHEGD$ | (1,1,1,1,2,1,2,2,1,1,2,1,2,2) |
| (40) | $S \rightarrow BIHEGDGHGHG$ | (3,1,3,2,3,1,1,2,1,1,2) |
| (41) | $S \rightarrow CKCFHDGHGLHDH$ | (1,1,1,2,1,1,1,1,6,1,2,1,1) |
| (42) | $A \rightarrow aA$ | $l(A1)=l(a)+l(A2)$ |
| (43) | $A \rightarrow a$ | $l(A)=l(a)$ |
| (44) | $B \rightarrow bB$ | $l(B1)=l(b)+l(B2)$ |
| (45) | $B \rightarrow b$ | $l(B)=l(b)$ |
| (46) | $C \rightarrow cC$ | $l(C1)=l(c)+l(C2)$ |
| (47) | $C \rightarrow c$ | $l(C)=l(c)$ |
| (48) | $D \rightarrow dD$ | $l(D1)=l(d)+l(D2)$ |
| (49) | $D \rightarrow d$ | $l(D)=l(d)$ |
| (50) | $E \rightarrow eE$ | $l(E1)=l(e)+l(E2)$ |
| (51) | $E \rightarrow e$ | $l(E)=l(e)$ |
| (52) | $F \rightarrow fF$ | $l(F1)=l(f)+l(F2)$ |
| (53) | $F \rightarrow f$ | $l(F)=l(f)$ |
| (54) | $G \rightarrow gG$ | $l(G1)=l(g)+l(G2)$ |
| (55) | $G \rightarrow g$ | $l(G)=l(g)$ |
| (56) | $H \rightarrow hH$ | $l(H1)=l(h)+l(H2)$ |
| (57) | $H \rightarrow h$ | $l(H)=l(h)$ |
| (58) | $I \rightarrow iI$ | $l(I1)=l(i)+l(I2)$ |
| (59) | $I \rightarrow i$ | $l(I)=l(i)$ |
| (60) | $J \rightarrow jJ$ | $l(J1)=l(j)+l(J2)$ |
| (61) | $J \rightarrow j$ | $l(J)=l(j)$ |
| (62) | $K \rightarrow kK$ | $l(K1)=l(k)+l(K2)$ |

| | | |
|---|---|---|
| (63) | $K \rightarrow k$ | $l(K)=l(k)$ |
| (64) | $L \rightarrow lL$ | $l(L1)=l(l)+l(L2)$ |
| (65) | $L \rightarrow l$ | $l(L)=l(l)$ |
| (66) | $M \rightarrow mM$ | $l(M1)=l(m)+l(M2)$ |
| (67) | $M \rightarrow m$ | $l(M)=l(m)$ |

where (1,1,1,1,1,1,1,2,1,1,1,2,1,1,3,1) is a shorthand for the inherited attributes whose meaning should be clearly understood from the previous examples.

This attributed grammar has 67 productions, a more than 90% reduction from the nonattributed grammar which requires 720 productions for 91% correct recognition. There are only 13 nonterminal symbols in this attributed grammar, which is equal to the number of terminal symbols. The nonattributed grammar has 681 nonterminals. The number of nonterminal symbols will not increase in this attributed grammar and the number of productions will increase at most by one for each additional input string. We can also expand the inherited attribute into a set of numbers. For example, we may let $L(A) = \{2, 3, 4\}$, which means the length of nonterminal symbol $A$ can be 2, 3 or 4. This will greatly increase the flexibility in some applications.

## 4.3 Error-Correcting Parsing of Attributed Seismic Grammar

A modified Earley's parsing algorithm is used for our attributed context-free seismic grammars. We assume that substitution, insertion and deletion of terminal symbols are allowed, but no substitution, insertion or deletion of nonterminal symbol is permitted. This means

the length of the local segment is variable, even local noise is tolerable, but the whole local segment can not be deleted entirely. The local segment means a segment of identical terminal symbols. The item of this parsing algorithm has the form $[A \rightarrow \alpha \cdot \beta, \eta, \xi, i]$ where $\eta$ is a counter for local syntactic deformation which accumulates the total cost of substitution of terminal symbols. $\xi$ is used for two different purposes. When $A \neq S$, $\xi$ is used as synthesized attribute of $A$. On the other hand, if $A = S$ then $\xi$ is used as a counter for semantic deformation which records the total length variation of nonterminal symbols, and $i$ is the same pointer as a conventional Earley's parser. A parsing algorithm for expanded attributed grammar using length attribute has been proposed by Leung (1982). As usual, we don't need an expanded grammar. All the deformations are examined during the parsing while errors are recorded in appropriate counters. The parsing algorithm is shown in the following.

*Algorithm 4.2* Minimum-Distance Error-Correcting Parsing Algorithm for Attributed Context-Free Seismic Grammar.

*Input:* An attributed seismic grammar $G = (V_N, V_T, P, S)$ and an input string $y = b_1 b_2 \ldots b_m$ in $V_T^*$.

*Output:* The parse lists $I_0, I_1, \ldots, I_m$, and decision whether $y$ is accepted by the grammar $G$ together with the syntacic and semantic deformation distances.

*Method:*

(1) Set $j = 0$. Add $[S \rightarrow \cdot \alpha, 0, 0, 0]$ to $I_j$ if $S \rightarrow \alpha$ is a production in $P$.

(2) Repeat step (3) and (4) until no new items can be added to $I_j$.

(3) If $[A \rightarrow \alpha \cdot B\beta, \eta, \xi, i]$ is in $I_j$, and $B \rightarrow \gamma$ is a production in $P$, then add item $[B \rightarrow \cdot \gamma, 0, 0, j]$ to $I_j$.

(4) (a) If $[A \rightarrow \alpha \cdot, \eta_2, \xi_2, i]$ is in $I_j$ and $[A \rightarrow a \cdot A, \eta_1, \xi_1, k]$ is in $I_i$, then add an item $[A \rightarrow aA \cdot, \eta_1+\eta_2, \xi_1+\xi_2, k]$ to $I_j$. (There is no need to check collision here, since there will be no other item of the form $[A \rightarrow aA \cdot, \eta, \xi, k]$ in $I_j$.)

(b) If $[A \rightarrow \alpha \cdot, \eta_2, \xi_2, i]$ in $I_j$ and $[S \rightarrow \beta \cdot A\gamma, \eta_1, \xi_1, k]$ is in $I_i$, then add an item $[S \rightarrow \beta A \cdot \gamma, \eta_1+\eta_2, \xi_1+(L(A)-\xi_2), k]$ to $I_j$, where $L(A)$ is the inherited attributed of the nonterminal symbol $A$.

(5) If $j=m$, go to step (7); otherwise $j=j+1$.

(6) For each item $[A \rightarrow \cdot a\beta, \eta, \xi, i]$ in $I_{j-1}$ add $[A \rightarrow a \cdot \beta, \eta+S(a,b_j), \xi+l(b_j), i]$ to $I_j$, where $l(b_j)$ is the synthesized attribute of $b_j$. For simplicity, we may let $l(b_j) = 1$ for all $j$. $S(a,b_j)$ is substitution cost, and $S(a,b_j) = 0$ when $a = b_j$. Go to (2).

(7) If item $[S \rightarrow \alpha \cdot, \eta, \xi, 0]$ is in $I_m$, then string $y$ is accepted by grammar $G$ where $\eta$ is the syntactic deformation distance and $\xi$ is the semantic deformation distance; otherwise, string $y$ is not accepted by grammar $G$. Exit.

A flow chart of this parsing algorithm is given in Figure 4.2. It is noted that (1) The parse extraction is straightforward once the first S-production is identified, therefore we do not include the parse extraction algorithm. This is obvious. Since we use attributes, the syntactic part will be much simpler than that of a nonattributed (context-free) grammar. (2) Deformation of any type on terminal symbols will be accepted. For a simple example, the string '*aaadgggegggggggegeeg*' will be accepted by our seismic grammar with no error; the string

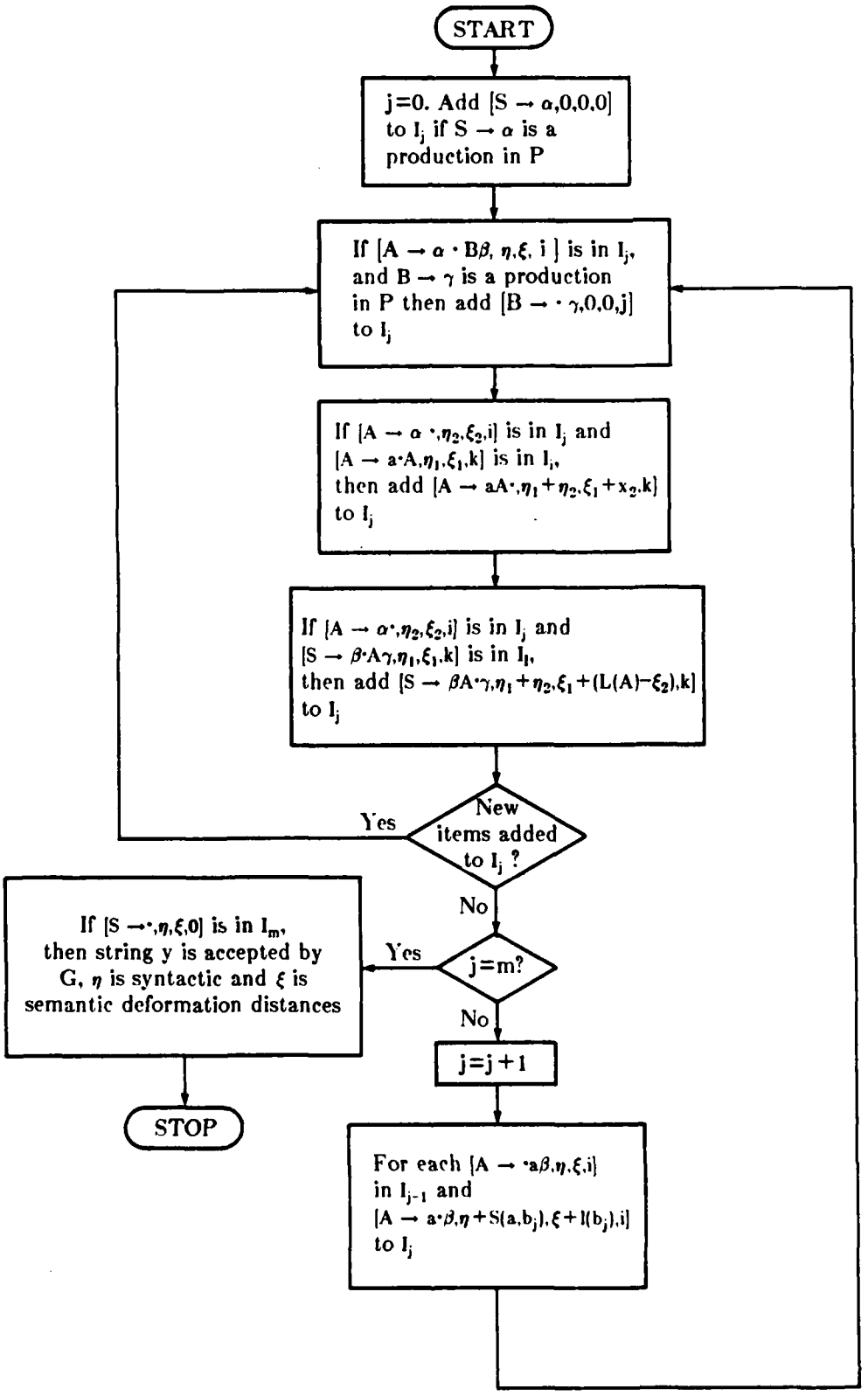



Figure 4.2 A flow chart of the parsing algorithm (Algorithm 4.2).

'*aadgggeg*...' will be accepted with semantic error of one unit length on '*A*'; and the string '*abadgggeg*...' will also be accepted with a syntactic substitution error $S(a,b)$.

The time complexity of Algorithm 4.2 is $O(n^2)$ where $n$ is the length of the input string, since each item list $I_j$ takes time $O(j)$ to complete. However, since we only considered substitution error in the seismic recognition problem in Section 3.5, a simplified version of Algorithm 4.2, i.e., Algorithm 4.4, can be applied. This special parser is faster than Algorithm 3.2. The experimental results are given in Section 4.5. The question about how much advantage we can take by using attributes depends on the selection and characters of the training samples. If the training samples are very much alike, then there are great possibilities that less syntactic rules are needed; instead, attributes will be used to distinguish between different patterns. An attributed grammar can also be constructed manually based on the knowledge about pattern sources. This may sometimes be a great advantage.

## 4.4 Stochastic Attributed Grammar and Parsing for Seismic Analysis

Although we do not know the probability distribution of the training samples at this moment, it is possible to estimate it if more samples are available. If the probability distribution of the training samples is known, then we can infer the production probability using the algorithm described in Lee and Fu (1972b). Therefore, we also include a parsing algorithm for stochastic attributed seismic grammar in this section. A stochastic version of the attributed grammar shown in

Section 4.2 can be formulated as follows. First, a probability is associated with each production. Second, a probability distribution is associated with all the possible attributes. For example, if originally $L(A) = \{3, 4, 5\}$, now it may become $L(A) = \{(3, 0.25), (4, 0.5), (5, 0.25)\}$, where $0.25 = Prob\{L(A)=3\}$. Finally, probabilities instead of costs are used to characterize substitution transformations. The probability associated with each $S$-production will be the probability of occurrence of the training string which contributes to that production.

The parsing algorithm of stochastic attributed seismic grammar is very similar to Algorithm 4.2 except for the following changes. First, $\eta$ is now the probability of syntactic substitution deformation. Second, $\xi$ is still used as a synthesized attribute of $A$ when $A \neq S$, however, when $A = S$, $\xi$ will be the probability of semantic deformations.

*Algorithm 4.3* Error-Correcting Parsing Algorithm for Stochastic Attributed Seismic Grammar

*Input:* An attributed seismic grammar $G = (V_N, V_T, P, S)$ and an input string $y = b_1 b_2 \ldots b_m$ in $V_T^*$.

*Output:* The parse lists $I_0, I_1, \ldots, I_m$, and decision whether $y$ is accepted by the grammar $G$ together with the syntactic and semantic deformation probabilities.

*Method:*

(1) Set $j = 0$. Add $[S \rightarrow \cdot\, \alpha, 1, 1, 0]$ to $I_j$ if $S \rightarrow \alpha$ is a production in $P$.

(2) Repeat step (3) and (4) until no new items can be added to $I_j$.

(3) If $[A \rightarrow \alpha \cdot B\beta, \eta, \xi, i]$ is in $I_j$, and $B \rightarrow \gamma$ is a production in $P$, then add item $[B \rightarrow \cdot\, \gamma, 1, 1, j]$ to $I_j$.

(4) (a) If $[A \rightarrow \alpha \cdot , \eta_2, \xi_2, i]$ is in $I_j$ and $[A \rightarrow \alpha \cdot A, \eta_1, \xi_1, k]$ is in $I_i$, then add an item $[A \rightarrow \alpha A \cdot , \eta_1 \cdot \eta_2, \xi_1 + \xi_2, k]$ to $I_j$. (There is no need to check collision here, since there will be no other item of the form $[A \rightarrow \alpha A \cdot , \eta, \xi, k]$ in $I_j$.)

(b) If $[A \rightarrow \alpha \cdot , \eta_2, \xi_2, i]$ in $I_j$ and $[S \rightarrow \beta \cdot A\gamma, \eta_1, \xi_1 k]$ is in $I_i$, then add an item $[S \rightarrow \beta A \cdot \gamma, \eta_1 \cdot \eta_2, \xi_1 \cdot Prob\{\xi_2\}, k]$ to $I_j$, where $L(A)$ is the inherited attributed of the nonterminal symbol $A$.

(5) If $j = m$, go to step (7); otherwise $j = j + 1$.

(6) For each item $[A \rightarrow \cdot \alpha\beta, \eta, \xi, i]$ in $I_{j-1}$ add $[A \rightarrow \alpha \cdot \beta, \eta \cdot P_S(b_j \mid \alpha), \xi + l(b_j), i]$ to $I_j$, where $l(b_j)$ is the synthesized attribute of $b_j$. For simplicity, we may let $l(b_j) = 1$ for all $j$. $P_S(b_j \mid \alpha)$ is substitution probability. Go to (2).

(7) If item $[S \rightarrow \alpha \cdot , \eta, \xi, 0]$ is in $I_m$, then string $y$ is accepted by grammar $G$ where $\eta$ is the syntactic deformation probability and $\xi$ is the semantic deformation probability; otherwise, string $y$ is not accepted by grammar $G$. Exit.

A flow chart of this parsing algorithm is given in Figure 4.3. Due to the error-correcting characteristics there may be more than one item of the form $[S \rightarrow \alpha \cdot , \eta, \xi, 0]$ in $I_m$. In that case, a decision should be made based on $\eta$ and $\xi$. Weights can be assigned to $\eta$ and $\xi$. Nevertheless, this is a rather subjective judgement, and is always a problem when using both syntactic and semantic informations.

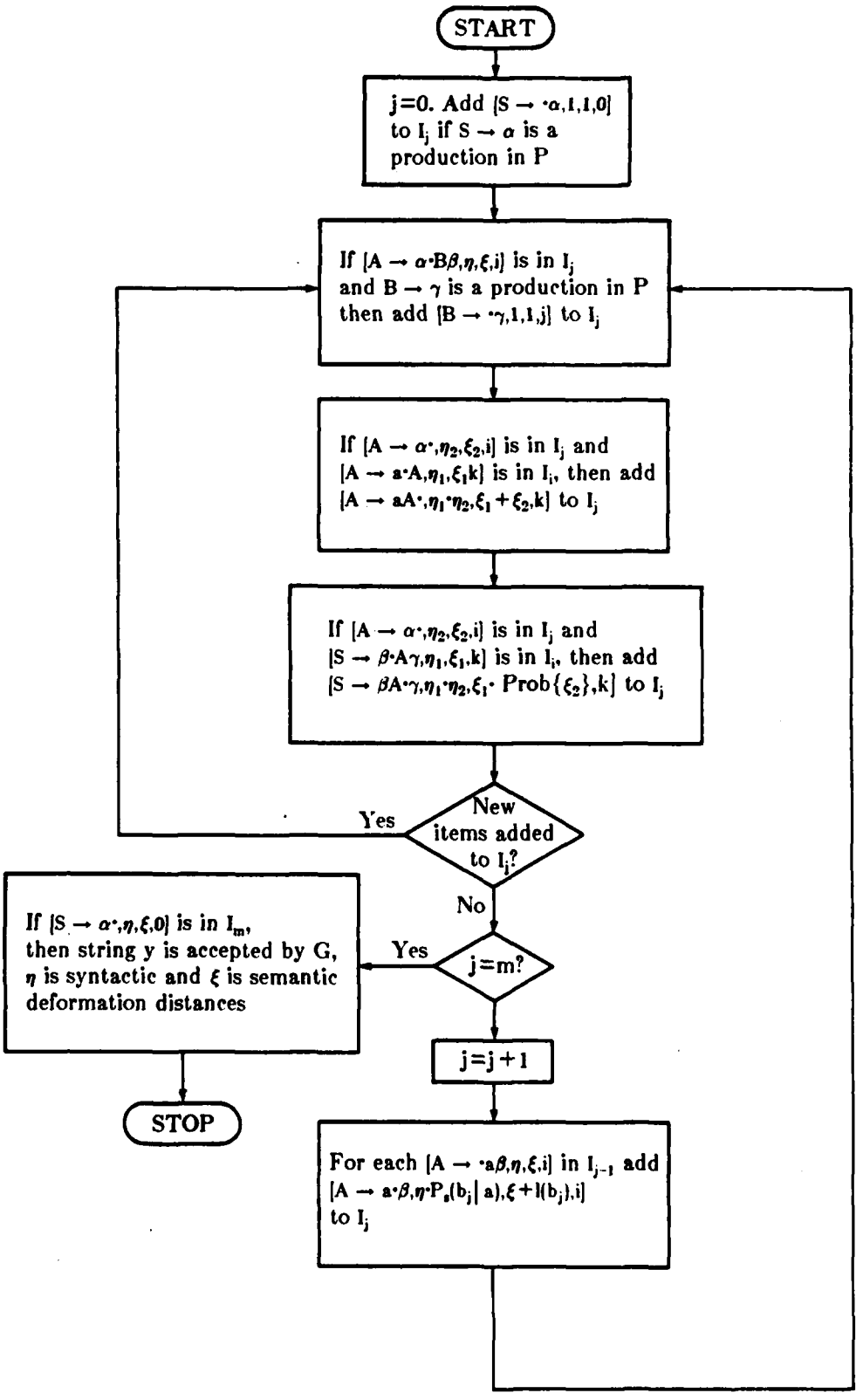



Figure 4.3 A flow chart of the parsing algorithm (Algorithm 4.3).

## 4.5 Experimental Results and Discussion

In this chapter we have shown an attributed seismic grammar which has only 67 productions and 13 nonterminal symbols compared to the 720 productions and 681 nonterminal symbols of a nonattributed finite-state grammar. An error-correcting parser (Algoritm 4.2) is also proposed for this attributed grammar. Since the error-correcting parser of Algorithm 3.2 considered only the substitution error, a simplified version of Algorithm 4.2 which ignores the length variation can be used to greatly increase the processing speed. This is shown in Algorithm 4.4.

*Algorithm 4.4* Top-Down No-Backtrack Error-Correcting Parsing Algorithm for Attributed Seismic Grammar.

*Input:* An attributed seismic grammar $G = (V_N, V_T, P, S)$ and an input string $y = b_1 b_2 \ldots b_m$ in $V_T^*$.

*Output:* The minimum distance between $y$ and $L(G)$ where only substitution error is considered.

*Method:*

(1) Set $N$ = the number of $S$-productions, min-distance = a sufficiently large number.

(2) Set $i = 1$.

(3) The $i$th $S$-production has the form $S_i \rightarrow A_{i1} A_{i2} \cdots A_{iM_i}$, where $M_i$ is the number of nonterminals at the right-hand side of the $i$th $S$-production, $A_{ij} \in V_N$, $1 \leq j \leq M_i$.

(4) Set dist = 0, $k = 1$, $l = 1$.

(5)(a) If $k > \sum_{p=1}^{l} L(A_{ip})$, then $l=l+1$.

(b) Apply production $A_{il} \rightarrow a_{il} A_{il}$ and compute dist = dist + $S(a_{il},b_{lk})$. $k=k+1$. Note that there is one-to-one correspondence between $A_{il}$ and $a_{il}$, $a_{il} \in V_T$.

(6) If $k \leq m$, go to step (5).

(7) If dist < min-distanct then min-distance = dist.

(8) $i=i+1$. If $i \leq N$ go to (3); otherwise min-distance is the minimum distance between $y$ and $L(G)$. Exit.

A flow chart of this parsing algorithm is given in Figure 4.4. A parse of $y$ can be constructed by tracing the productions used in Step (3) and (5)(b). If the length variation is to be considered then the item lists will contain a large number of items, and consequently the computation will be slow. However, Algorithm 3.2 is unable to even consider the length variation.

The recognition results and computation time for recognizing one string are given in Table 4.1. While both attributed cfg and nonattributed fsg show 91% correct recognition, the average computation time for one string is 0.11 second using attributed seismic grammar and is 2.55 second using nonattributed finite-state grammar. This is because the finite-state seismic grammar has a large number of production rules and nonterminal symbols. A large table must be maintained and searching is very time-consuming. Although a special-purpose hardware can be built to speedup the computation, it is slow for a sequential computer. Algorithm 4.2 is also time-consuming for a general context-free grammar. However, the seismic grammar in Section 4.2 is a very special cfg, and the application of the production rules is

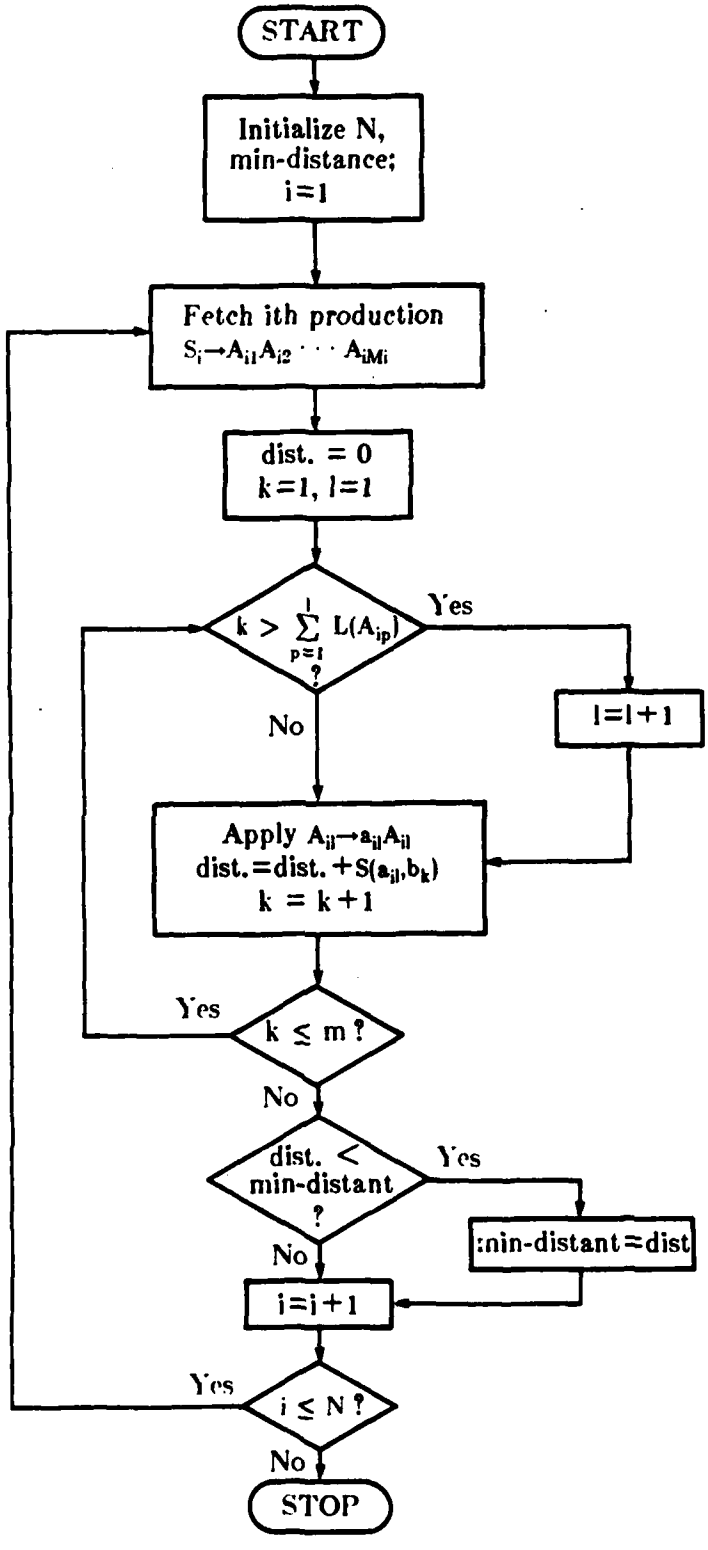



Figure 4.4 A flow chart of the parsing algorithm (Algorithm 4.4).

TABLE 4.1

**The recognition results, computation time, and memory used for seismic recognition using an attributed context-free grammar and a nonattributed finite-state grammar.**
(Time is for one string)

| | Accurate Rate | Average Time (sec) | memory used (bytes) |
|---|---|---|---|
| Attributed cfg | 91% | 0.11 | 41360 |
| Non-attributed fsg | 91% | 2.55 | 72804 |

very straightforward. The actual storage used in computer is also given in Table 4.1.

We mentioned earlier that substitution, insertion and deletion of terminal symbols are allowed but no substitution, insertion or deletion of nonterminal symbol is permitted. As a matter of fact, substitution of nonterminal symbols can be attained in terms of substitution of terminal symbols. Therefore, only insertion or deletion of nonterminal symbols is not allowed. The reason is that if the training samples are well selected, the grammar should be able to recognize any reasonable strings. If the test string needs insertion or deletion of nonterminal symbols in order to be accepted, it is either severely distorted or missing some string segments. If insertion and deletion of nonterminal symbols are to be considered then this becomes a structural-deformation problem (Tsai and Fu, 1979). We can add insertion and deletion error transformations in Step(6) of Algoritm 4.2 as we did in Algorithm 2.4. This will make the algorithm more complicated. A distance threshold should be imposed to eliminate unrealistic parses so that the item list will not become unmanagable.

# CHAPTER V

# VLSI ARCHITECTURES FOR SYNTACTIC SEISMIC PATTERN RECOGNITION

## 5.1 Introduction

Some computational algorithms, for example, matrix multiplication and inversion in numeric computation and string matching in non-numeric computation, are very time-consuming so that an efficient implementation is usually not feasible and economical. However, this situation has been changed due to the advances in hardware technology, i.e., the development of high-speed, high-density and low-cost electronic devices. Hardware implementation (particularly parallel and/or pipeline processing) of software algorithm has become an affordable solution to increase the processing speed because the cost of hardware is decreasing. Advance in VLSI technology makes it possible to pack more components into one chip at a lower price than ever before (Mead and Conway, 1980). This revolutionary impact stimulates considerable interest to develop parallel algorithms for VLSI implementation and build special-purpose chips for specific applications (Kung, 1979, 1980). A whole book (Bowen and Brown, 1982) has been devoted to VLSI systems design for digital signal processing. Many computers and processors have been developed for signal processing. The recent trend is to

use attached signal processors, e.g., Lincoln Laboratory Fast Digital Processor (FDP) and Data Genaral AP/130 array processor, instead of supercomputers as ILLIAC-IV and Advanced Scientific Processor (ASC) (Bowen and Brown, 1982). More specialized applications for matrix multiplication, convolution and solving linear equations can be found in Kung (1979, 1982), Kulkarni and Yen (1982), Hwang and Cheng (1981). A recent example of special-purpose VLSI architecture is an integrated multiprocessing array for time warping pattern matching which is used in speech recognition (Ackland, Weste and Burr, 1981). Pattern matching is the most time-consuming stage in speech recognition especially when the dictionary is large. Using parallel processing improves the speed 200 times faster, therefore make the real-time application possible.

Like dynamic time warping, all the string distance computation and string matching are time consuming. Hardware implementation has been proposed by Okuda, Tanaka and Kasai (1976) for computing Levenshtein distance even before VLSI technology is available. They used delay circuits to implement insertion, deletion and substitution weights.

We propose in this chapter a VLSI architecture for seismic classification using syntactic approach, which includes feature extraction, primitive recognition and string matching. Our string matching implementation is more complicated than Okuda, et al.'s, where different weights are assigned to different symbols in our case. This special-purpose processor is designed to be attached to a host computer, for example, a minicomputer as shown in Figure 5.1, therefore it works like a perihperal processor. Three systolic arrays are proposed

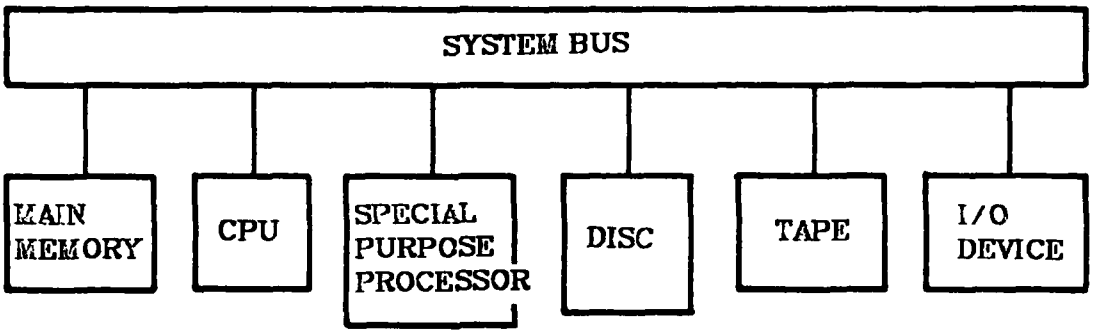



Figure 5.1 The special-purpose processor is attached to a host computer as a peripheral processor.

to perform feature extraction, primitive recognition and string matching respectively. Several memory units are required for holding the intermediate results and for data setup. Figure 5.2 shows the architecture of our special-purpose processor. All these three systolic arrays perform in time $O(1)$, i.e., results can be produced at a constant rate provided that input data are supplied properly in a pipelined fashion. The formations of input data are given in Figure 5.3 where ($a$) is for feature extraction, ($b$) is for primitive recognition and ($c$) is for string matching. Section 5.2 discusses VLSI architectures for feature extraction. Section 5.3 discusses VLSI architectures for primitive recognition. Section 5.4 discusses VLSI architectures for string matching. Section 5.5 shows some simulation results and performance verification. Section 5.6 gives the concluding remarks.

## 5.2 VLSI Architectures for Feature Extraction

The systolic array for feature extraction is linearly connected as shown in Figure 5.4. The input data, which are the digitized and quantized signal waveform coded in binary form, are stored in seperate memeory modules in a skewed format as shown in Figure 5.3($a$) and Figure 5.4($a$). Each memory module is delay by one unit time, i.e., time required to process one data element, from left to right. Each memory module contains a sequence of words, i.e., discrete signal points and is connected to a processing element (PE) of the systolic array. The data are transferred into the PE's bit by bit, and all the memory modules are read parallelly. Two features, zero-crossing count and sum of absolute magnitudes are computed. Absolute sum instead

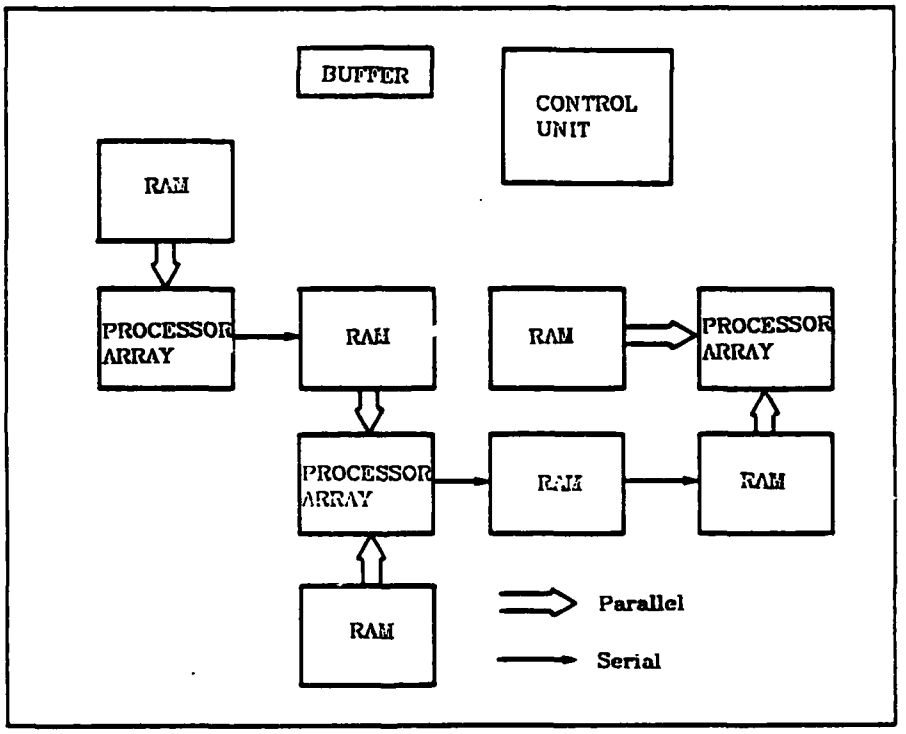



Figure 5.2 The internal architecture of the special- purpose processor.

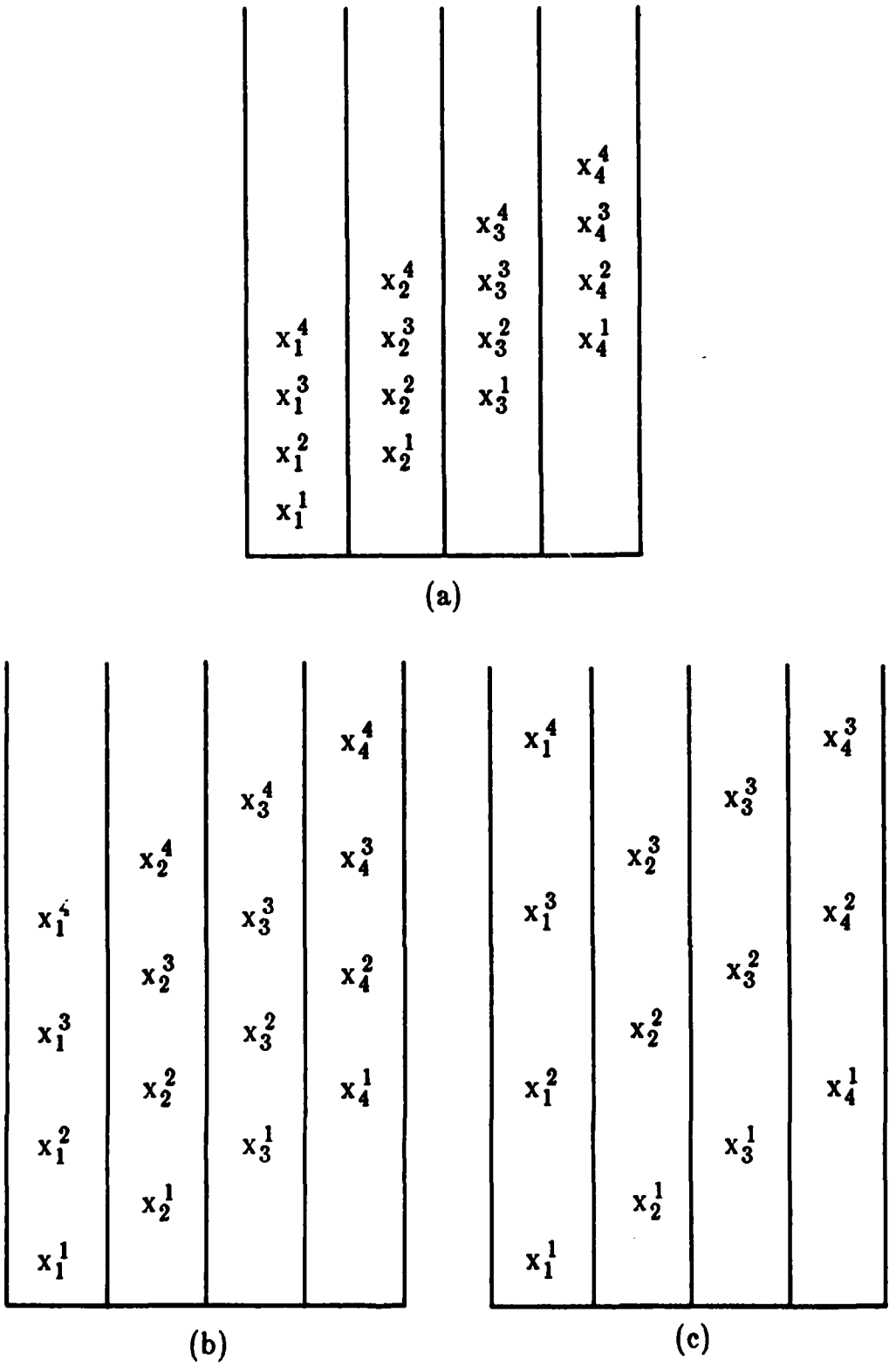
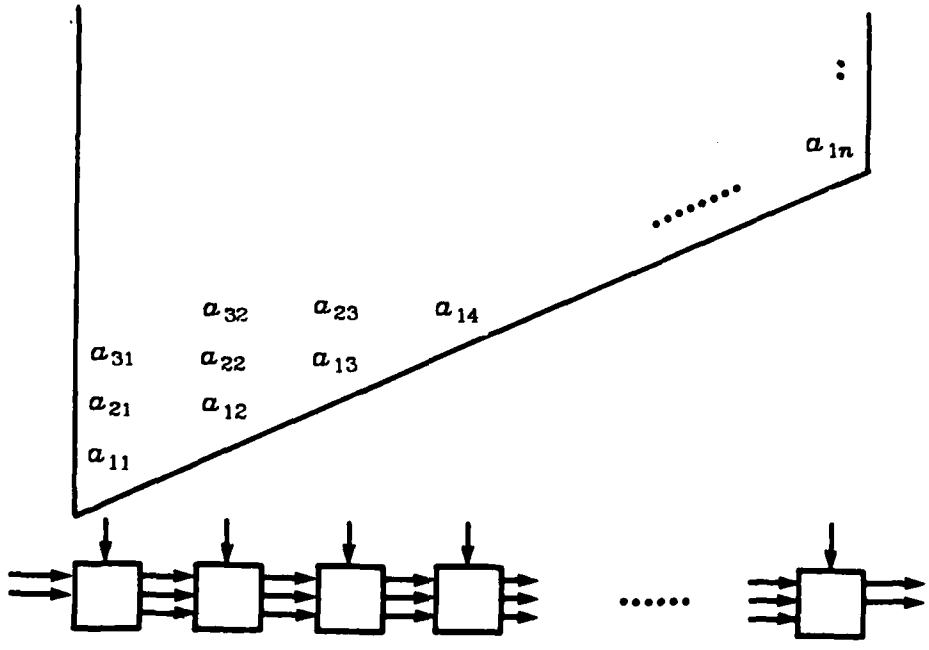
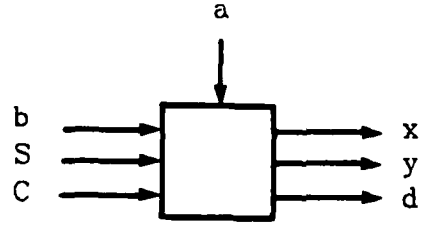



Figure 5.3 Data setup for (a) feature extraction, (b) primitive recognition and (c) string matching.




(a)




$y \leftarrow S + |a|$

$d \leftarrow C + (sgn(a) \oplus sgn(b))$

$x \leftarrow a$

(b)

Figure 5.4 Processor array, data movement and operations of each processor for feature extraction.

of log energy is used here for the simplicity of implementation. Logarithmic function can be approximated by taking a series expansion (see Ackland, et al., 1981). Zero-crossing is detected by checking the signs of every two consecutive points. Any sign change is counted as one zero-crossing. An exclusive-OR circuit is used for detection of sign change. Figure 5.4(*b*) shows the operation of each PE. The internal structures are given in Figure 5.5. All the $n$ PE's compute the two features simultaneously and pass the partial results to the next PE's. Each general-purpose register A, B, C, E and S is 16-bit long. The micro-operations of each PE are as follows.

(1) (a) Transfer (serially) input data into Register A from external storage.

(b) Transfer (serially) input data into Register B from Register A of the left PE.

(c) Transfer (serially) partial result into Register C from Register C of the left PE.

(d) Transfer (serially) partial result into Register S from Register S of the left PE.

(e) $C \leftarrow C + (sgn(A) + sgn(B))$.

(2) $E \leftarrow |A|$.

(3) $S \leftarrow S + E$.

Steps (1)(a) to (1)(e) can be executed in parallel, therefore can be completed in 16 machine cycles. Step (2) and step (3) can each be completed in one machine cycle. The entire operations (1), (2) and (3) take 18 machine cycles to complete. The time for each processor to complete its entire operations, i.e. 18 machine cycles here, is call a unit

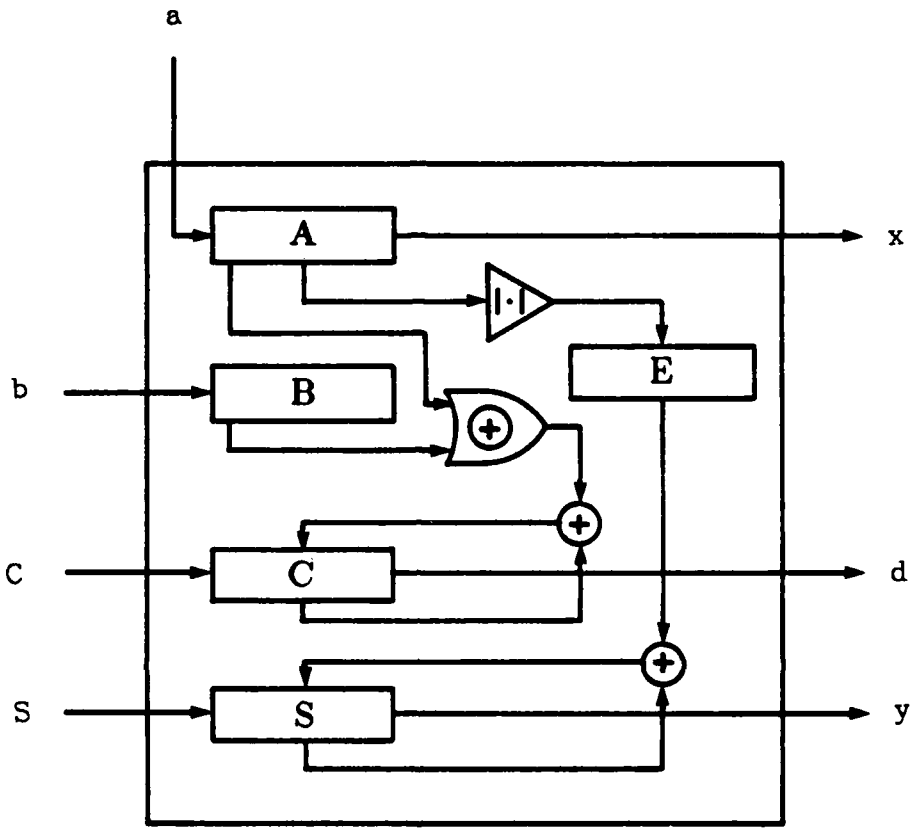



Figure 5.5 The internal structure of the processor for feature extraction.

time. Although memory cycle is slower than machine (procesor) cycle, each memory fetch can take time as long as 18 machine cycles. Therefore data input can keep up with processor speed. Suppose that input data are fed in properly, then after $n$ unit times, where $n$ is the number of data points in one segment, the feature of the first segment will emerge from the end of the systolic array. There will be a set of features (of one segment) coming out every unit time thereafter. Therefore with the systolic array reaching steady state, each segment only takes 1 unit times, i.e., 18 machine cycles, to complete the computation. With a uniprocessor, each segment will take $O(n)$ computations and comparisons. The speedup is $n$, which is equal to the number of processors.

### 5.3 VLSI Architectures for Primitive Recognition

In the primitive recognition problem, we compute the distance between the unknown feature vector and the reference vector, for example, mean vector, of each cluster (primitive), and then assign the unknown feature vector to the cluster of the minimum distance. This procedure can be divided into two steps; first, compute the distances between the unknown vector and the reference vectors, and then select the smallest distance. We use a processor array, which contains 'compute' processors, for distance computation and a processor array, which contains Suppose there are $l$ primitives; each primitive $i$ has a reference feature vector $[m_1^i, m_2^i, ..., m_k^i]$ where $k$ is the total number of features. A processor array of $l$ by $k$ which performs the distance computation is shown in Figure 5.6. The reference vectors of the

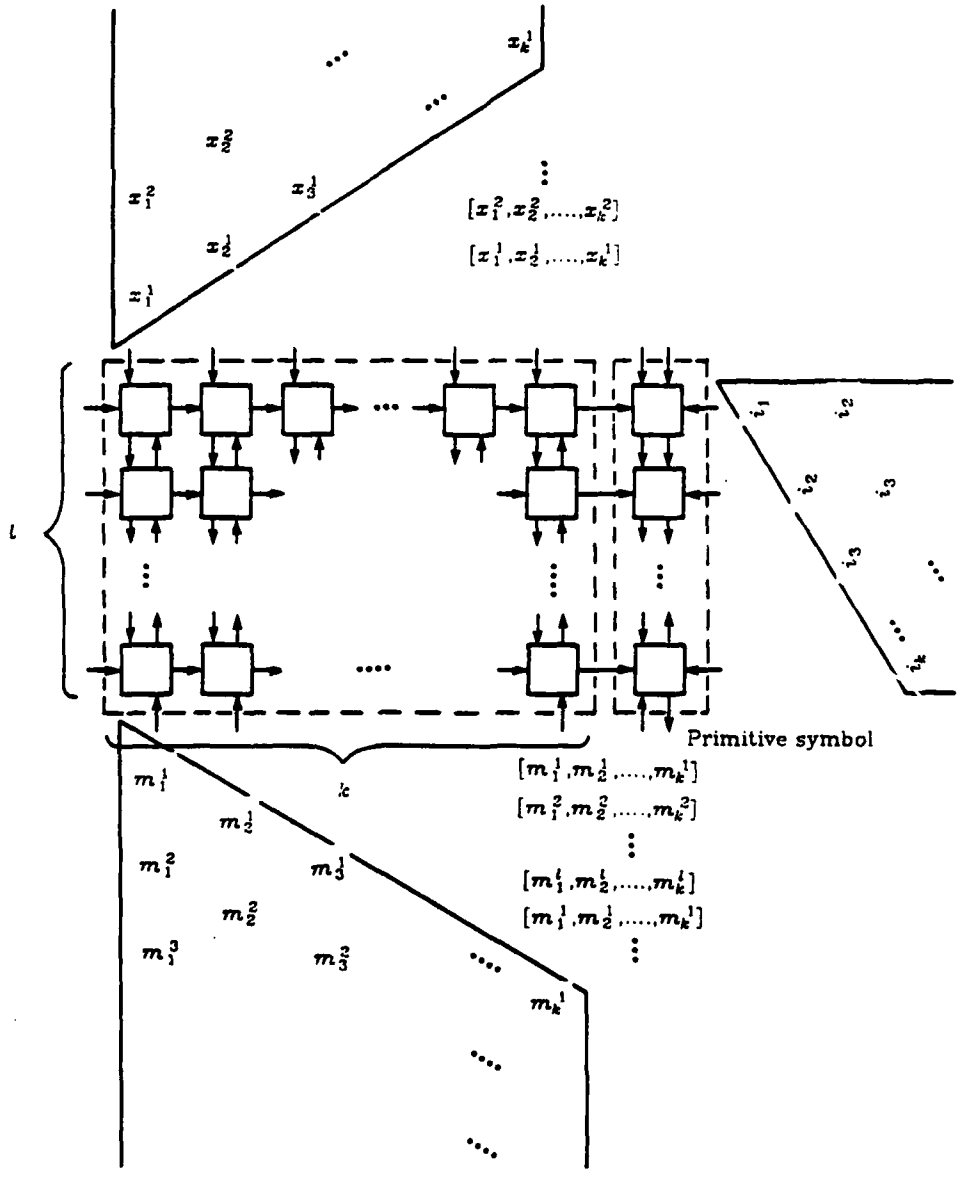



Figure 5.6 Processor arrays and data movement for primitive recognition.

primitives enter from the bottom and move up while the unknown feature vectors enter from the top and move down. The partial sums move from left to right. The data must be properly skewed as shown in Figure 5.6 and Figure 5.3($b$). Since the two data streams move in opposite direction, they must be separated by one unit time which is shown by one space in Figure 5.6; otherwise, some data will just pass instead of meeting each other.

The unknown feature vectors are assumed to come in continuously. The reference vectors must also repeat their cycles continuously, i.e., with the first primitive vector coming right after the $l$th primitive vector. After initiation, the feature vectors will be delayed for $l-1$ unit times so that the first feature vector and the first primitive vector will meet at the first row of the processor array. The sum, which is equal to zero initially, will be the distance at the end of computation. The functional diagram of each 'compute' processor is shown in Figure 5.7(a), where $x$ is a component of the unknown feature vector, $u$ is a component of the primitive vector and $a$ is the partial sum. For simplicity, we use the absolute-value distance here. Euclidean distance computation will take more space and time.

The internal structure and data movement are shown in Figure 5.8(a). Each 'compute' processor contains an arithmetic and logic unit (ALU), and four 16-bit registers A, B, U and X. The micro-operations are shown as follows.

(1) (a) Transfer data (serially) into register X from the above PE.

(b) Transfer data (serially) into register U from the lower PE.

(c) Transfer partial sum (serially) into register A from the left PE.

(2) B ← X - U.

(3) B ← |B|.

(4) A ← A + B.

Step (1) takes 16 clock cycles to transfer one word of 16 bits; step (2), (3) and (4) takes 1 clock cycle each. The entire operations take 19 clock cycles. The unit time here is 19 clock cycles.

After computation of the corresponding components between the reference vector and the unknown feature vector, the partial sum moves to the right. When the partial sum passes the $k$th processor of the first row, the output will be the distance between vectors $[x_1^1, x_2^1, \ldots, x_k^1]$ and $[m_1^1, m_2^1, \ldots, m_k^1]$, then it enters the rightmost column of processors, which are the 'compare' processors. Since the data streams are seperated by one unit time, the processors on alternate diagonals (from lower left to upper right) are idle. When vector $[x_1^1, x_2^1, \ldots, x_k^1]$ enters the second row of the processor array, it will meet vector $[m_1^2, m_2^2, \ldots, m_k^2]$. When vector $[x_1^1, x_2^1, \ldots, x_k^1]$ enters the third row, it will meet vector $[m_1^3, m_2^3, \ldots, m_k^3]$; meanwhile, vector $[x_1^2, x_2^2, \ldots, x_k^2]$ will meet vector $[m_1^2, m_2^2, \ldots, m_k^2]$ at row one. We can see from the above and Figure 5.6 that vector $[x_1^1, x_2^1, \ldots, x_k^1]$ is compared with the reference vectors in the sequence 1, 2, ..., $l$, vector $[x_1^2, x_2^2, \ldots, x_k^2]$ is compared with the reference vectors in the sequence 2, 3, ..., $l$, 1, and so forth. These operations are overlapped, i.e., pipelined, in a way that every processor is doing part of the computation and pass the data and results to the neighbor processors.

The functional diagram of the 'compare' processor is shown in Figure 5.7(b) where $a$ is the minimum distance computed so far with primitive identifier $c$, $b$ is the distance just computed and $d$ is the corresponding primitive identifier input externally. The internal structure and data movement are shown in Figure 5.8(b). Each 'compare' processor contains an ALU, two 8-bit registers B, D and two 16-bit registers A, C. The micro-operations are as follows.

(1) (a) Transfer partial sum (serially) into register A from C of the above PE.

(b) Transfer partial sum (serially) into register C from the left PE.

(c) Transfer primitive identifier (serially) into register B from D of the above PE.

(d) Transfer primitive identifier (serially) into register D from external input.

(2) E ← A - C.

(3) If $a < c$ then {C ← A; D ← B}.

Step (1) takes 16 cycles to complete, step (2) takes 1 and step (3) takes 1. These three steps take 18 cycles, which is 1 cycle shorter than the 'compute' processor, therefore the 'compare' processor must be idle for one cycle in order to synchronize with the 'compute' processor. The 'compare' processors compare the current distance coming from the left with the distance coming from the above, and pass the smaller one to the lower processor. Primitive identifiers are fed in from the right in a similar format as those for data streams. The identifier streams should be delayed for $l+k-1$ unit times so that the first identifier $i_1$ enters the first 'compare' processor at the same time as

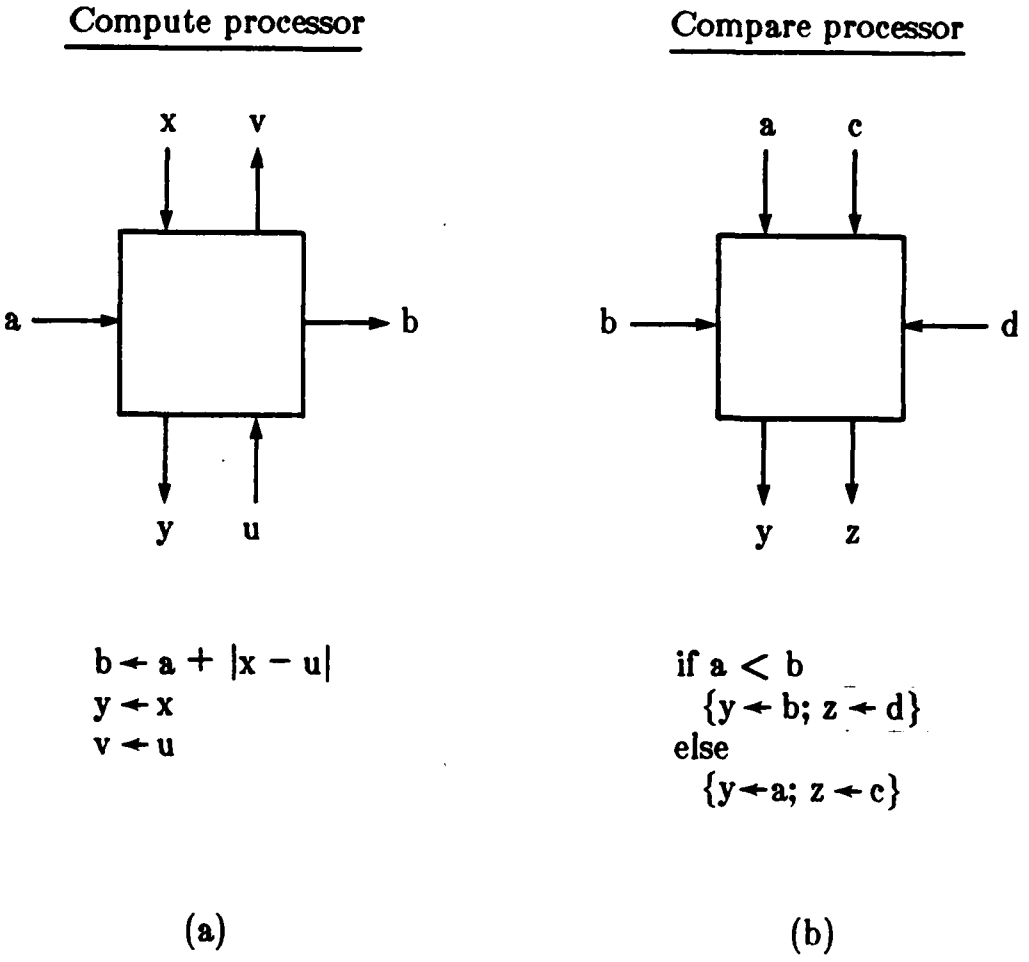



Figure 5.7 Data flow and operations of each (a) 'compute' processor and (b) 'compare' processor.

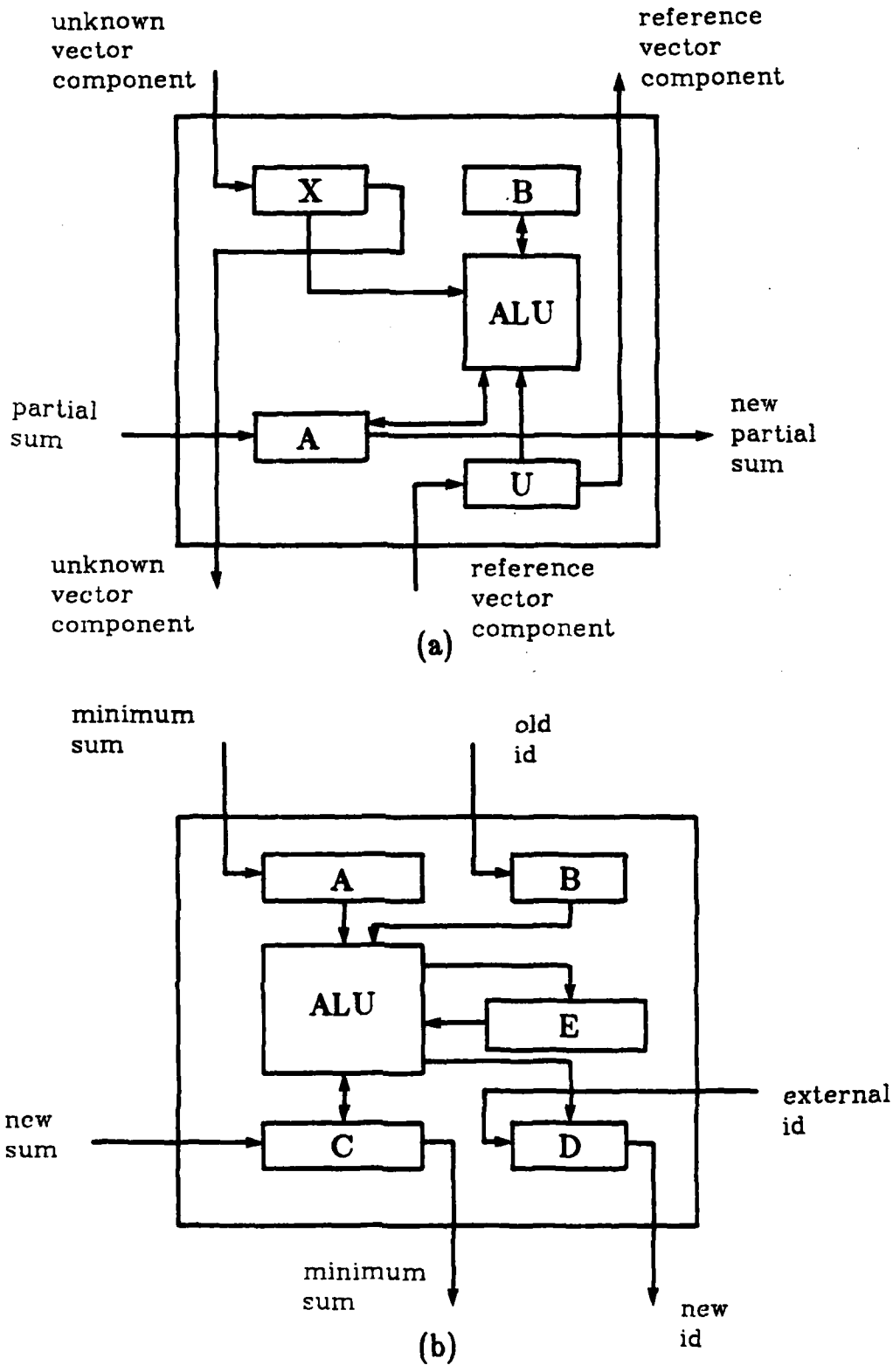



Figure 5.8 Internal structure and register transfer of (a) 'compute' and (b) 'compare' processors.

the distance between $[x_1^1, x_2^1, ..., x_k^1]$ and $[m_1^1, m_2^1, ..., m_k^1]$. In order to assign right identifier to right distance, the identifier streams must be arranged as shown in Figure 5.6.

With a uniprocessor, the primitive recognition procedure of one feature vector will take $l \times k$ computations and $l-1$ comparisons in our present example. With the processor array of Figure 5.6, the primitive recognition procedure of a single feature vector needs $l \times k + 1$ unit times. However, a processor array is not designed for the processing of one single datum, instead, it is for a stream of data. In that case, a new result will come out every 2 unit times in Figure 5.6. Given $l$ reference vectors and a feature vectors of dimension $k$, the array processor will take 2 unit times to get one result in steady state, while a uniprocessor takes $O(l \times k)$ time to complete the computation. The speedup is $l \times k / 2$. In Figure 5.6, the results contain both the minimum distance and the primitive identifier, therefore no other processing is required.

Primitive recognizer is essentially a vector pattern matcher. Therefore it can be used in many other applications, and can be used indepent of feature extraction and string matching.

## 5.4 VLSI Architectures for String Matching Based on Levenshtein Distance

Nonnumeric computation has become more important and demanded more hardware algorithms, i.e., algorithms specially designed for hardware implementations, and architectures recently due to the increasing applications in artificial intelligence, database, information retrieval, language translation, pattern recognition, etc.,

One of the most important categories in nonnumeric computation is string pattern matching. Character string matching is very important in information retrieval and dictionary look up (Hall and Dowling, 1980). The problem of string pattern matching can generally be classified into two kinds. We call them exact matching and approximate matching. For exact matcing, a single string is matched against a set of strings, usually this particular string is embeded as a substring of the reference strings. Hardware algorithms for exact matching has been proposed by Mukhopadhyay (1979), where the test pattern resides in an array of cells and the reference text is broadcasted to all the cells simultaneously character by character. Foster and Kung (1980) designed a VLSI chip for exact pattern matching with wild card capability, where the test pattern enters from one end and the reference text enters from the other end of the linear array. By constrast, for approximate matching, we want to find a string from a finite set of strings which approximately matches the test string. Certainly we will also find the string which exactly matches the test string if it does exist. A good survey of approximate string matching can be found in Hall and Dowling (1980). This section concentrates exclusively on approximate matching. Approximate string matching is based on the idea of insertion, deletion and substitution of terminal symbols. An application example of approximate string matching which cannot be performed by exact string matching is the string clustering problems, for example, in Lu and Fu (1978). Wagner and Fischer (1974) proposed a dynamic programming method for the computation. Okuda, Tanaka and Kasai (1976) proposed an algorithm and hardware implementation for garbled word recognition based on the Levenshtein Metric. We propose in this

section a VLSI architecture for approximate string matching. The distance measure is (weighted) Levenshtein distance using dynamic programming method. Although it is using the minimum-distance criterion in deterministic cases; it can be easily modified to the maximum-likelihood criterion in probabilistic cases.

Chiang and Fu (1979) studied several parallel architectures, namely, SIMD, dedicated SIMD and MIMD, for string and tree distance computation. Each node on the same diagonal of the dynamic programming matrix is computed simultaneously. The time complexity of these specific parallel systems is $O(n+m)$, where $n$ and $m$ are the lengthes of the two strings under comparison. Our system, differs from theirs in that we use a systolic array, i.e., a square array of PE's as in Ackland, et al. (1981) and pipelined data flow for the computation. Therefore we can obtain the results at a constant rate, i.e., one result after each unit time.

It is well-known that Levenshtein distance can be computed by dynamic programming. Therefore, it can be implemented by parallel processing on VLSI architectures. In this case, parallel computation and pipeline data flow are combined to process continuously a large amount of data at a very high speed. The dynamic programming algorithm recursively computes the optimal path from point (1,1) to $(m,n)$ based on its subpaths. In dynamic time warping, there are many slope constraints for selecting subpaths. Ackland et al. (1981) chose the simplest constraint, i.e.,

$$S_{i,j} = D_{i,j} + \min \left\{ S_{i-1,j}, S_{i-1,j-1}, S_{i,j-1} \right\}$$

where $D_{i,j} = |x_i - y_j|$, $x_i$, $y_j$ are feature vectors, $S_{i,j}$ is partial sum at point $(i,j)$. It will be much difficult to implement if they chose other slope constraints.

### 5.4.1 Levenshtein Distance

For Levenshtein distance, there are also many variations. The original Levenshtein distance where each insertion, deletion and substitution is counted as one error transformation is the easist to implement. We have developed a processor array for this computation. A portion of the dynamic programming diagram and its corresponding processor array is given in Figure 5.9. Each processor computes the partial sum

$$S_{i,j} = \min \begin{Bmatrix} S_{i-1,j}+1 \\ S_{i-1,j-1}+S(a_i,b_j) \\ S_{i,j-1}+1 \end{Bmatrix}$$

where $S(a_i,b_j) = 1$ if $a_i \neq b_j$; $S(a_i,b_j) = 0$ otherwise. The computation can be divided into three stages. The procedures are as follows.

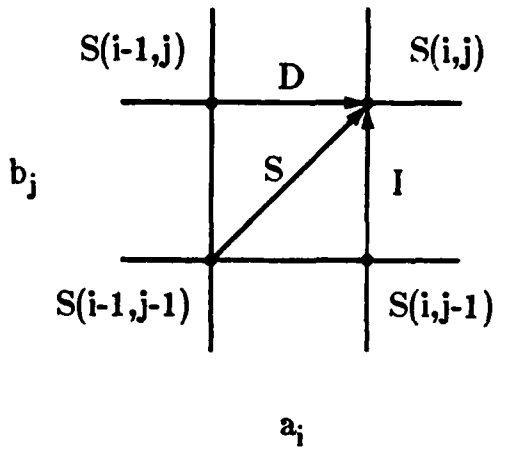
*Stage 1*

(1) (a) Transfer (serially) partial sum $S_{i-1,j-1}$ into D from the lower PE.

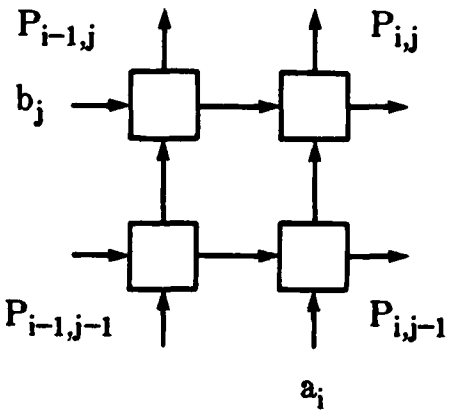
(b) Transfer (serially) primitive $a_i$ into X from the lower PE.

(c) Transfer (serially) primitive $b_j$ into Y from the left PE.

(d) Compare (serially) X with Y; output V = 0 if X = Y, V = 1 otherwise.




(a)




(b)

Figure 5.9 (a) Portions of dynamic programming diagram and (b) corresponding processor array.

(2) D ← D + V.

*Stage 2*

(1) (a) Transfer (serially) partial sum $S_{i-1,j}$ into B from the left PE.

(b) Transfer (serially) partial sum $S_{i,j-1}$ into C from the lower PE.

(c) Send (serially) partial sum $S_{i-1,j}$ to D of the above PE.

(d) Compare (serially) B with C, A ← min(B, C).

(e) Send (serailly) contents of X to X of the above PE.

(f) Send (serially) contents of Y to Y of the right PE.

(2) A ← A + 1.

(3) Compare (parallelly) A with D, R ← min(A, D).

*Stage 3*

(1) (a) Send (serially) partial sum R to B of the left PE.

(b) Send (serially) partial sum R to C of the above PE.

Stage 1 takes 17 clock cycles to complete (16 for step (1) and 1 for step (2)); stage 2 takes 18 (16 for step (1), 1 for step (2) and 1 for step (3)), and stage 3 takes 16. Figure 5.10 shows the internal structure and the operations of processor element $P_{i,j}$ at stage 1, 2 and 3. Each PE contains a set of registers, an ALU, a control unit and some other combinational logic. Registers A, B, C, D, V and R are general-purpose registers which are 16-bit long and connected to the ALU. Registers X and Y are 8-bit long, which are used to store primitives. In our seismic case, we have 13 primitives; therefore, 4 bits should be enough to represent them. In fact, 4 bits, which have 16 combinations, should be sufficient for many practical applications. However, in order to make our system more flexible and compatible with other systems which use ASCII code,

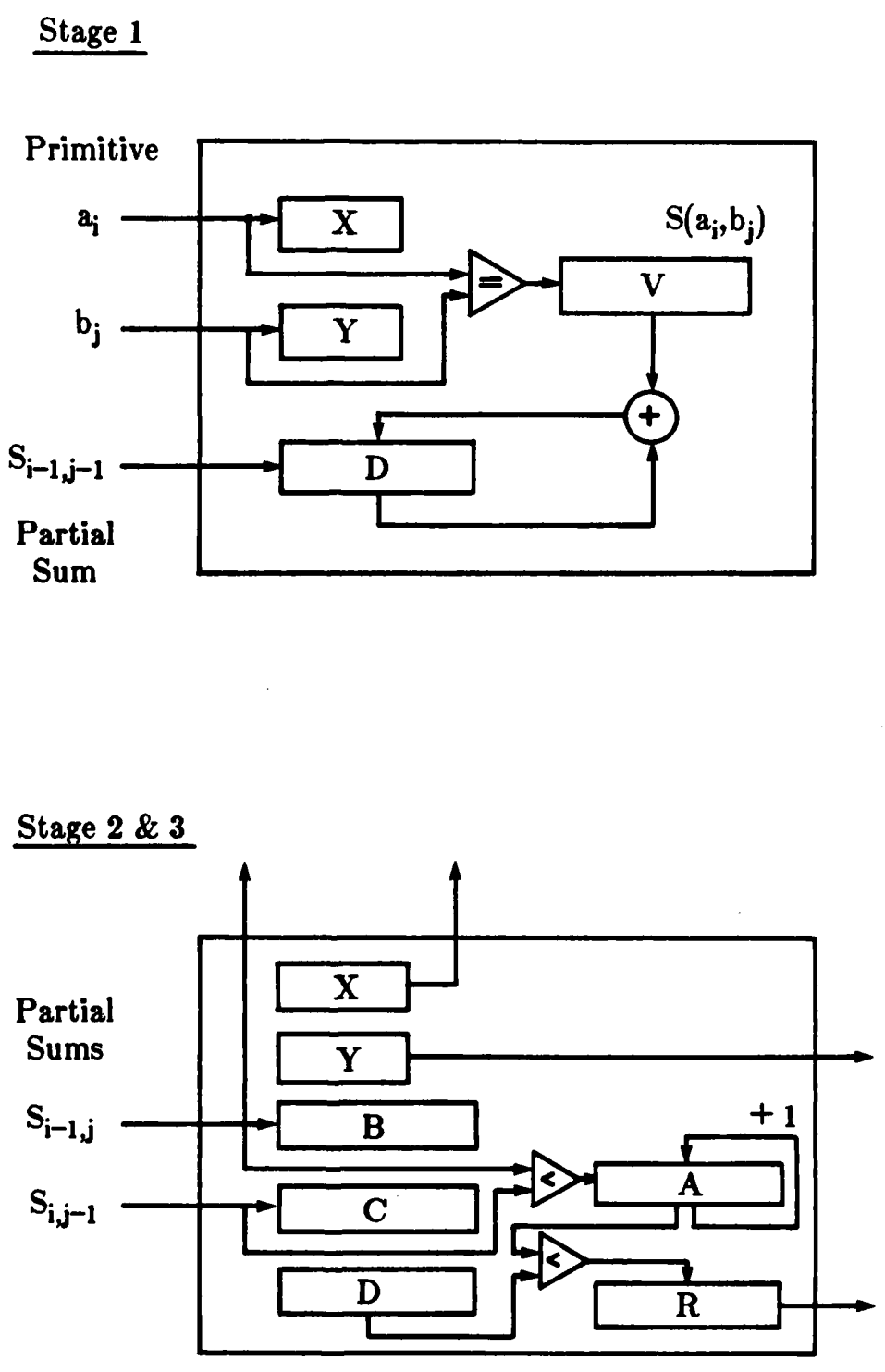



Figure 5.10 Internal structure and register transfer of PE $P_{i,j}$ at stage 1, 2 and 3.

we let registers X and Y hold 8 bits. This generalization will be able to recognize character strings where each character is in ASCII code, for example, A = '01000001', B = '01000010', C = '01000011', and so forth.

Figure 5.11 shows the data movement between 4 neighboring PE's shown in Figure 5.9. All the processors at the same diagonal performs the same computation as shown in Figure 5.11 and 5.12(a). This format will move forward one step every 18 clock cycles. Since each string only needs three diagonals at any time, the other processors can be used to compute distances of other strings. Therefore, data flow can be pipelined as shown in Figure 5.12(b). If we are matching a test string against a number of reference strings, the distance between the test string and the first reference string will emerge after $p \times 18$ clock cycles, where $p$ is the number of diagonals in the array. After that, there will be one string distance coming out every $3 \times 18 = 54$ clock cycles. Since stage 1 and 3 have no conflict, they can be ovelapped, i.e., one diagonal of the array can be used to perform stage 3 of one string and stage 1 of the next string at the same time, to increase the throughput.

The structure of processor array and data flow are shown in Figure 5.13. The reference strings enter from the left; the test string enters from the bottom. The test string must repeat itself continuously in order to compare with all the reference strings. Both test and reference strings must be properly skewed and separated as shown in Figure 5.13 so that they will arrive at the right processors at the right time. The bookkeeping and selection of minimum distance can be done by a special-purpose processor or the host computer. One practical problem is about the dimension of the processor array. The number of rows

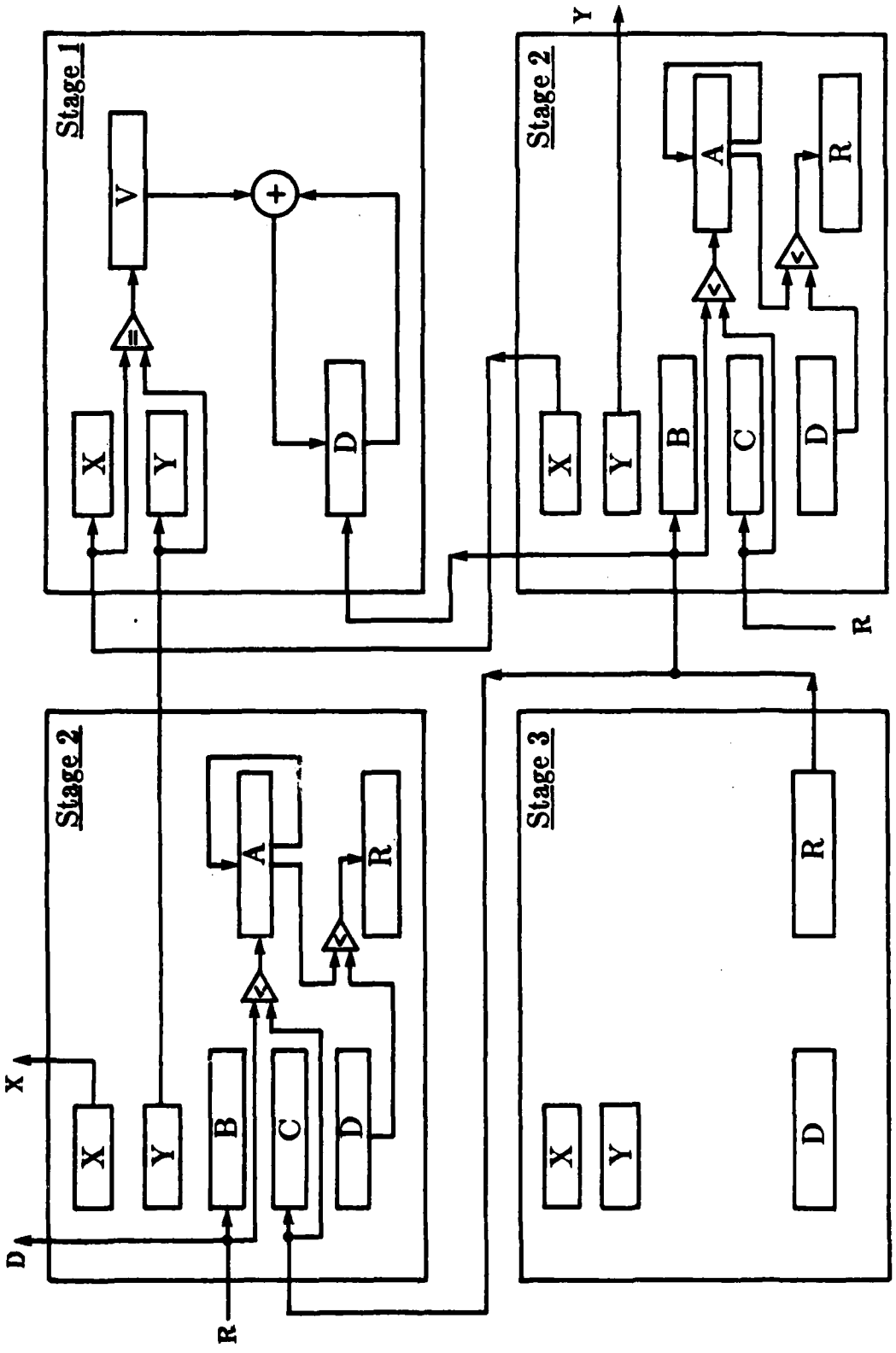
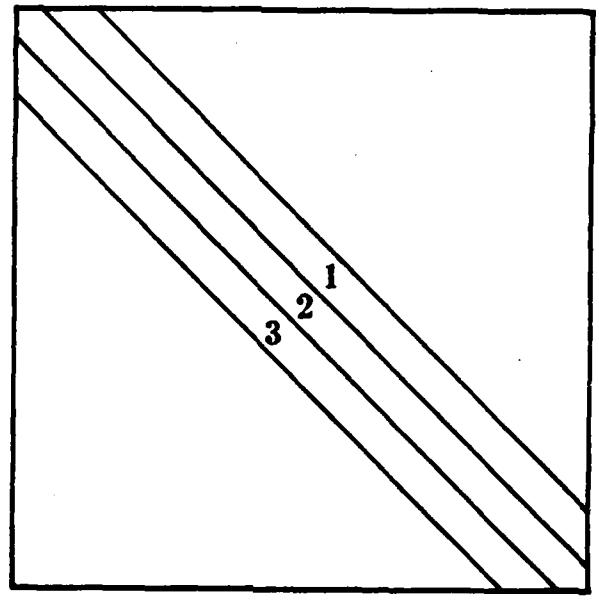
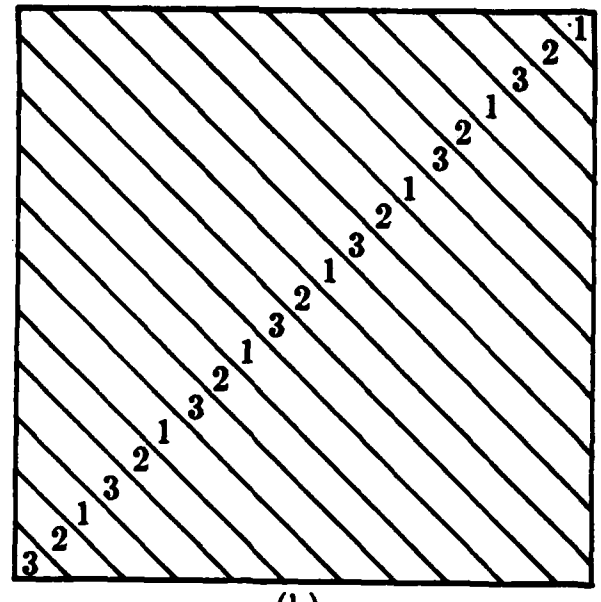



Figure 5.11 Data movement between PE's.




(a)




(b)

Figure 5.12 Processors at the same diagonal perform the same operation; three diagonals are required for one string (a), and strings can be pipelined (b).

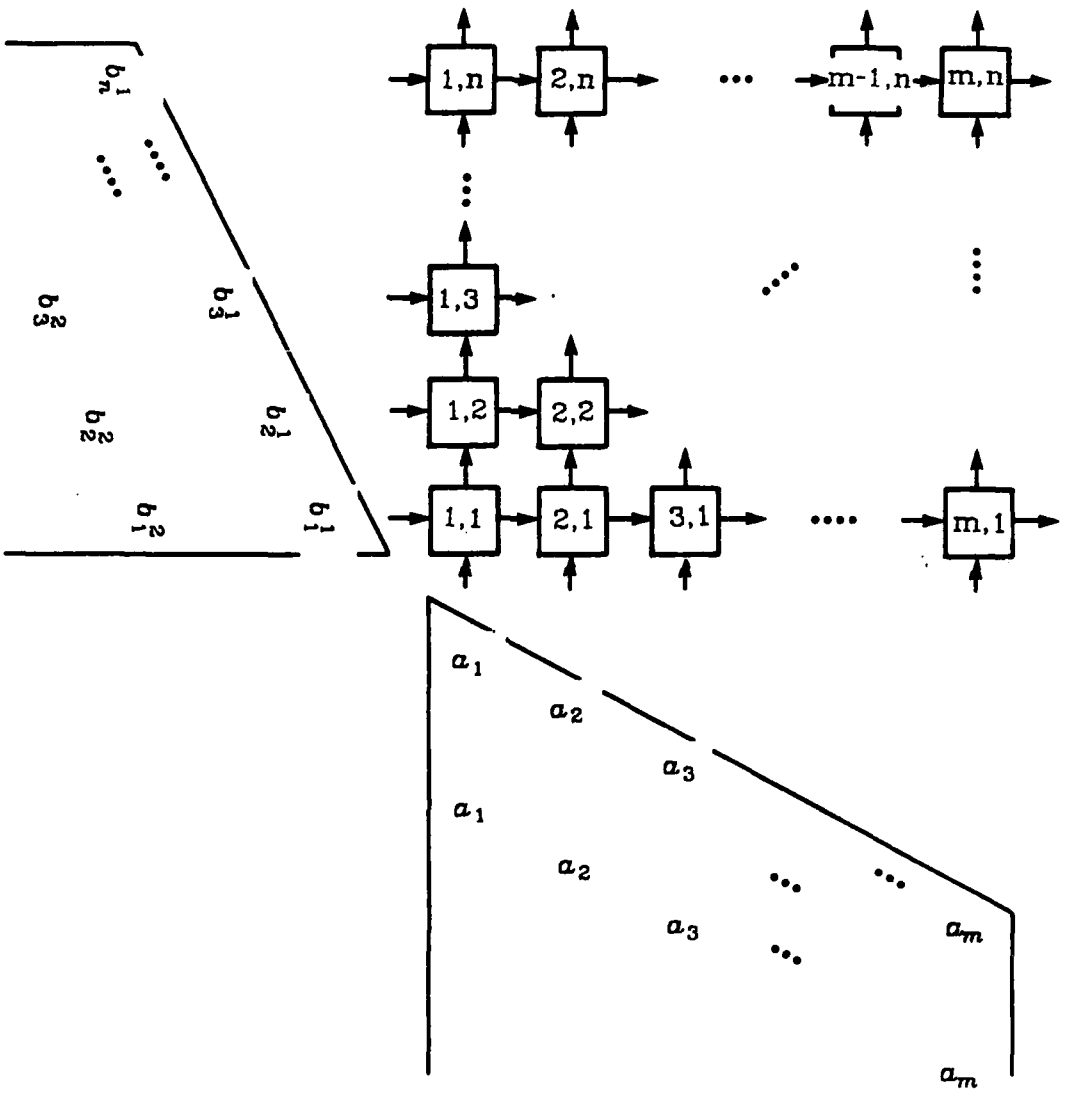



Figure 5.13 Processor array and data movement for computing Levenshtein distance.

can be set to the maximum length of the reference strings. Since the length of the test string is unknown, the number of column can be set arbitrarily. If a test string exceeds the array size, it should be handled by the host computer or preprocessor. Because the interruption of the regular computation pattern in a VLSI array will greatly reduce its efficiency. This situation can be kept to minimum by selecting a reasonablly large array size. A shorter string will be padded out with blank to make it equal to the array dimension.

Suppose both the reference and the test strings have length $l$. With a uniprocessor, the matching process for one unknown string will take $O(l \times l)$ unit operations. With the array processor, it only takes 3 unit times.

### 5.4.2 Weighted Levenshtein Distances

Since a weighted Levenshtein distance is usually more favorable in practical application, we now propose a VLSI architecture for its computation. The major problem here is to store all the weights in each processor, which must be easy to implement and fast for access. Fortunately, a programmable logic array (PLA) can be used (Mead and Conway, 1980). It is a special type of read-only memory, and easy to implement in a VLSI system. A simple example will illustrate how a PLA works. Figure 5.14 shows a simple weights table and its PLA implementation. A PLA consists of two parts, the left part is called the AND plane, the right part is called the OR plane. Input lines A, B have the combinations (0,0), (0,1), (1,0), (1,1) which represent the entries of the weight table. The output XYZ indicate the values of the entries, which range from 0 to 7. The circles indicate connections. Since we only have

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 1 | 3 |
| 1 | 5 | 7 |

| | X | Y | Z |
|---|---|---|---|
| 1 = | 0 | 0 | 1 |
| 3 = | 0 | 1 | 1 |
| 5 = | 1 | 0 | 1 |
| 7 = | 1 | 1 | 1 |

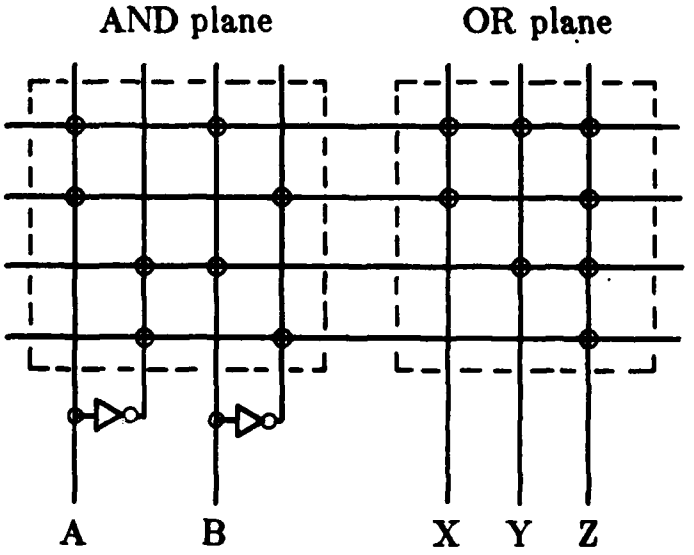



Figure 5.14 PLA implementation of a simple weight table.

13 primitives, 4 bits will be enough for discrimination. We take the 4 least-significant bits (LSB) from the primitives for our internal computation, for example, *a* = '0001', *b* = '0010', *c* = '0011', and so forth. We need more bits for recognition of character strings. Figure 5.15 shows the PLA implementation of weight table for substitution, insertion and deletion in our seismic case. There is an input register to the AND-plane and an output register from the OR-plane; both are 8-bit long. Register X contains primitive '*a*', and register Y contains primitive '*b*'; (*a*, *b*) is the entry of the weight table. Here the symbols X, Y, A and B are registers which should not be confused with those in Figure 5.14. The pair (X = *a*, Y = *b*) represents the substitution of '*b*' for '*a*'. The pair (X = *a*, Y = 0000) represents the deletion of '*a*'. The pair (X = 0000, Y = *b*) means the insertion of '*b*'. The access time is very fast, only two clock cycles; one is needed for input register, the other is for output register.

Except for the weight table, the computation procedure is similar to the previous one. The internal structure of the PE's is given in Figure 5.16. Each PE has an ALU, a PLA (with registers Q and S), a control unit, two 8-bit registers X, Y and three 16-bit registers B, C, and D. Register Z contains constant '0000' as symbol λ. The data movement is similar to that in Figure 5.11.

*Stage 1*

(1) (a) Transfer (serially) partial sum $S_{i-1,j-1}$ into D from the lower PE.

(b) Transfer (serially) primitive $a_i$ into X from the lower PE.

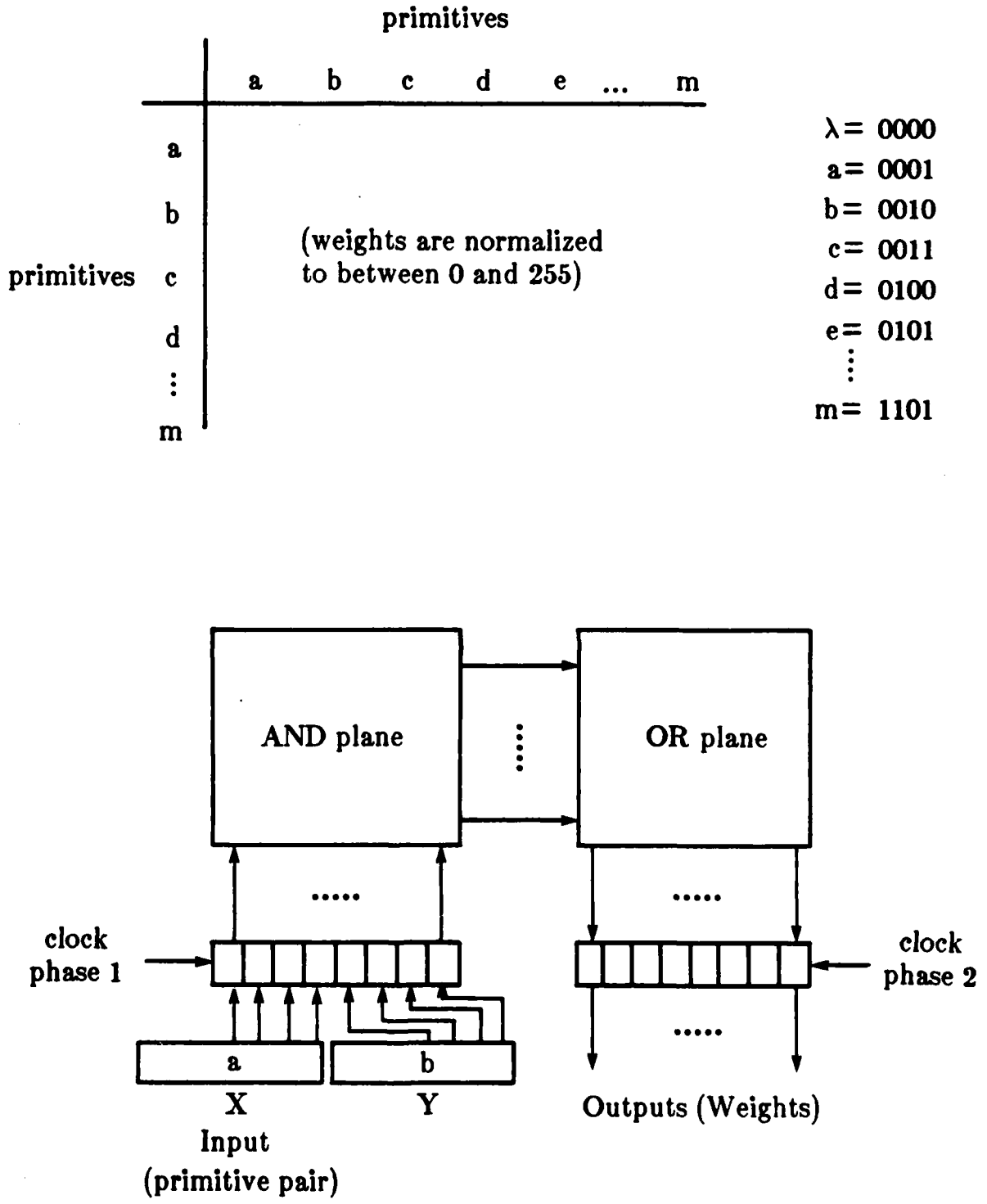



Figure 5.15 A PLA implementation of the weight table for seismic recognition.

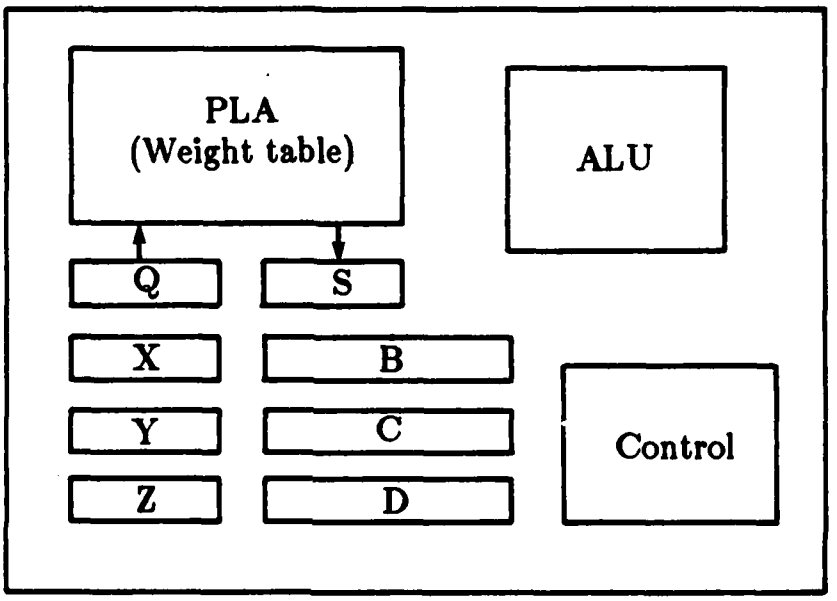



Figure 5.16 Internal structure of the PE for weighted string distance computation.

(c) Transfer (serially) primitive $b_j$ into Y from the left PE.

(2) Load (parallelly) the 4 LSB of X and Y into Q, output $S(a_i,b_j)$ in S.

(3) Compute D ← D + S.

(4) Load (parallelly) the 4 LSB of X and Z into Q, output $D(a_i)$ in S.

*Stage 2*

(1) (a) Transfer (serially) partial sum $S_{i-1,j}$ into B from the left PE.

(b) Transfer (serially) partial sum $S_{i,j-1}$ into C from the lower PE.

(c) Send (serially) partial sum $S_{i-1,j}$ to D of the above PE.

(d) Send (seraily) contents of X to X of the above PE.

(e) Send (serially) contents of Y to Y of the right PE.

(2) (a) Compute B ← B + S.

(b) Load (parallelly) the 4 LSB of Z and Y into Q, output $I(b_j)$ in S.

(3) (a) Compute C ← C + S.

(b) Compute B ← min(B, D).

(4) Compute D ← min(B, C).

*Stage 3*

(1) (a) Send (serially) partial sum in D to B of the right PE.

(b) Send (serially) partial sum in D to C of the above PE.

In Stage 1, Step (1) takes 16 cycles ((a), (b) and (c) operate in parallel), Step (2) takes 3 cycles (1 for loading, 2 for PLA reading), step (3) takes 1 cycle and Step (4) takes 3 cycles (same as Step (2)). In Stage 2, Step (1) takes 16 cycles, Step (2) takes 3 cycles ((a), (b) operate in parallel), Step (3) takes 2 cycles and Step (4) takes 2 cycles. Stage 3 takes 16 cycles ((a) and (b) both take 16 cycles and can be executed in parallel).

Therefore, Stage 1 takes 23 cycles, Stage 2 takes 23 cycles, and Stage 3 takes 16 cycles. As usual, stage 3 can be overlapped with stage 1 to save processing time. Due to the weight computation, this system takes longer time than the previous one.

## 5.5 Simulations and Performance Verification

Simulations have been performed for the three systolic arrays: feature extraction array, primitive recognition array and string matching array. The flow charts for the simulations are given in Appendix A. The same seismic data as those used in Section 3.5 are tested in the simulations. The results of the simulations are exactly the same as those of the sequential computer in Section 3.5. Therefore the design of the systolic arrays are correct and the operations are as expected. Step-by-step simulation results using sample seismic waveforms are given in Appendix B. The computation time in our simulation is shown in Table 5.1. The computation time using a sequential computer is also given for comparison. it is noted that the listed computation time is an average and approximate time which should be used for comparison only. Suppose that we are dealing with a large amout of data. Similar to the definition of speedup for multioperation computer in Kuck (1978), we define the theoretical speedup (TS) of a (systolic) processor array as

$$TS = \frac{\text{time interval between consecutive results using a sequential computer}}{\text{time interval between consecutive results using a processor array}}$$

TABLE 5.1

Computation time of sequential algorithm, simulated computation time for VLSI arrays using sequential computer, real speedups, theoretical speedups and speedup ratio. (All computations are for one seismic record or equivalent one string)

| | Sequential Algorithm (sec.) | Simulated VLSI arrays (sec.) | Real Speedup | Theoretical Speedup | Speedup Rate |
|---|---|---|---|---|---|
| Feature Extraction | 0.3 | 0.005 | 60 | 60 | 100% |
| Primitive Recognition | 0.1 | 0.006 | 17.4 | 19.5 | 89% |
| String Matching | 0.07 | 0.015 | 4.67 | 6.67 | 70% |

Therefore the TS for feature extraction is 60/1 = 60, for primitive recognition is 39/2 = 19.5 and for string matching is 20/3 = 6.67. The numerators are the numbers of operations for getting one result using a sequential computer, and the denominators are the time intervals between consecutive results for VLSI arrays as shown in the previous sections. Note that the TS for string matching in our experiment is 20/3 = 6.67 instead of 20×20/3 = 133. This is because we only consider substitution errors, therefore the number of operations is proportional to string length, i.e., 20. If insertion and deletion errors are to be considered, then the whole dynamic programming matrix as shown in Figure 2.1 should be considered. In the seismic recognition problem the size of the matrix is 20×20.

The real speedup in our simulations for feature extraction is approximately the same as the maximum theoretical speedup. This is due to the simple structure and data flow of the linearly connected systolic array. The real speedup (17.4) for primitive recognition is slightly less than the TS (19.5), which is 89% of the TS. The reason for this is the increasing complexity of array structure and data flow. More time is spent on data movement. The real speedup (4.67) for string matching is also less than the TS (6.67), which is 70% of the TS. This is because the array structure and data flow are even more complicated. The increasing complexity can be seen from the the designs in previous sections. The theoretical speedup is the upper bound where the real speedup in simulation is a function of the computations performed and the underlying architectures.

The simulations are performed on a sequential computer (VAX 11/780). In order to compare the simulation results with the results in

Chapter III we use the same high level languages (C, Fortran and Pascal). Therefore, there are many overhead in language translation and program execution. These are some of the reasons for low speedup. Another reason is data movement which can be performed in parallel with the computation in VLSI arrays, but can not be done in a sequential computer. One can not accurately simulate the VLSI system even using an assembly language. Since most systolic arrays are hardwired, i.e., unprogrammable, there is no instruction decoding or memory fetch and storage for each instruction. Besides, the parallelism can not be fully simulated on a sequential computer. The real computation speeds of the proposed VLSI arrays when fabricated should stay close to the analytical results as shown in the previous sections, i.e., 1 unit time for feature extraction, 2 unit times for primitive recognition and 3 unit times for string matcing using WLD.

We would like to consider some problems about actual implementation and give some examples about the performance of our proposed system. In Section 5.2 we assumed that the length of the linear systolic array is the same as the number of points in each segment. Although a linear array can be expanded easily, it is sometimes necessary to use small array to process data of larger size. For example, in the seismic recognition problem, the number of points in each segment is 60. We can use a linear array consisting of 60 PE's, or we can use less PE's, for example, 20 PE's. The implementation using 20 PE's is shown in Figure 5.17, where the data points in each segment are folded into three rows. This will take three unit times to compute the features for each segment. Suppose there are 20 PE's with machine cycle 200 ns, then the time required for feature extraction of 2,000 segments (after it reaches

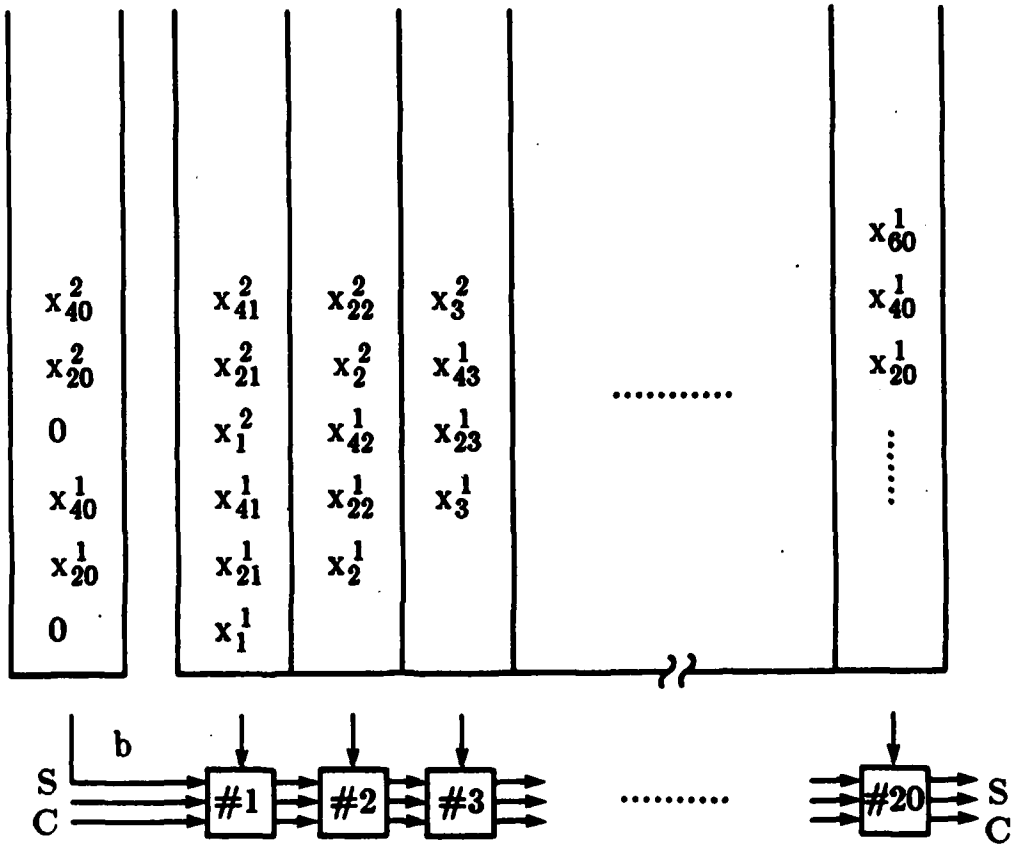



Figure 5.17 An implementation of feature extraction with 20 PE's and 60 points in each segment.

steady-state) is equal to 3 × 18 × 2000 × 200 ns = 21.6 ms. The time for reaching steady-state is 20 unit times, which is equal to 20 × 18 cycles × 200 ns/cycle = 72 $\mu$s. Since the processing speed of feature extraction (18 cycles) is faster than that of primitive recognition (19 cycles), the output of the former can be used directly by the latter. Recall that the input data for primitive recognition are interleaved by one space (Figure 5.3(*b*)). The feature vector of the next segment is not needed until after 2 × 19 cycles = 38 cycles. Therefore 30 PE's can be used for feature extraction and produce a feature vector every 2 × 18 cycles = 36 cycles. Because 30 PE's take 2 unit times to produce a result and each unit time is equal to 18 cycles. These two operations can be executed in parallel to save a half of the total processing time.

Consider string matching using Levenshtein distance, the comparison of one test string with each reference string takes 3 × 18 × 200 ns = 10.8 $\mu$s. With one hundred reference strings, it takes 1.08 ms to classify each test string, and each test string is executed sequentially. Using a systolic array it is possible to make real-time string matching. Our system can match approximately 90,000 strings per second (10.8 $\mu$s for one string).

We assume in the previous discussion that all strings, test and reference, have the same length. This is not true in many other applications. Reference strings are different in length; dimensions of processor array can not fit exactly the string size. It is required to make processor array larger than the string size and pad the string with blank at the end. If we let the weight of insertion, deletion and substitution of blanks be zero, then we can solve the problem of length variety and still maintain the regular, synchronous data flow pattern.

Long strings and larger array size do not degrade the steady-state throughput. As usual, results can be obtained every $3\times18 = 54$ cycles. It only takes longer time, i.e., $p\times18$ cycles where $p$ is the number of diagonals, to reach steady-state. Usually the time to reach steady-state is negligible compared with the total processing time.

The system bus as shown in Figure 5.1 is similar to the Unibus of DEC PDP-11 (Kuck, 1978). The Unibus has a maximum data rate of $4\times10^7$ bits/sec operating in an interlocked way, i.e., the sender waits until the receiver acknowledges receipt of a word before sending another word. In our experiment, each seismic record has 1200 points, and each point is coded into a 16-bit binary number. Therefore, each seismic record needs $16\times1200 = 19200$ bits of storage. It is easy to see that the system bus can transmit one seismic record from disc to special-purpose processor in 0.48 ms. However, the typical operating speed of magnetic disc is from $2.4\times10^5$ bits/sec to $1.2\times10^7$ bits/sec (Stone, 1980). Therefore the actual time for sending a seismic record from disc to special-purpose processor is from 80 ms to 1.6 ms. The output from the special-purpose processor is the classification results, which use one word (16 bits) for each seismic record to indicate class membership. The transmission time is 0.4 $\mu$s for one record.

## 5.6 Concluding Remarks

We have proposed special-purpose array processors for seismic signal classification, which can be attached to a general-purpose computer as shown in Figure 5.1. The host computer can retrieve any intermediate data from a special-purpose processor and store them in its own

storage, as well as send data to any memory unit of the special-purpose processor. For example, the host computer can retrieve and store the string representation of the signals for display or for later use. The host computer can also use any one of the systolic arrays, for example, feature extraction array, only.

The design correctness and speedup have been verified by simulations in Section 5.5. From the simulation results it is safe to predict that the real speedup of the fabricated VLSI processor arrays will be close to the theoretical speedup. Computer-aided design has greatly reduce the design cost (Swerling, 1982). The cost/performance ratio of special-purpose processors will eventually be justified.

Recently, VLSI architectures have been applied to syntactic pattern recognition and to implement parallel computation. Guibas, et al. (1979) proposed two VLSI arrays for the implementation of combinatorial algorithms, one is for a subset of dynamic programming problems, i.e., optimal parenthesization problems which include context-free language recognition, the other is for transitive closure problems which include finite-state language recognition. Based on the array structure of Guibas, et al., Chu and Fu (1981) proposed VLSI architectures for finite-state language recognition and context-free language recognition using CYK's algorithm. Chiang and Fu (1982) also proposed a VLSI systems for context-free language recognition using Earley's algorithm. Ackland et al. (1981) developed a VLSI systems to implement dynamic time warping for spoken word recognition. Our string matcher can be applied to any problem where the Levenshtein distance computation is required. It can be used for string matching in our seismic recognition, for character string matching in information

retrieval (Hall and Dowling, 1980) or for pattern matching in shape analysis if the object can be represented by a string, ,for example, using chain codes (see Fu, 1982). Our primitive recognizer can also be applied to any minimum-distance recognition problem and vector pattern matching.

# CHAPTER VI

# SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

## 6.1 Summary

We have studied the application of syntactic pattern recognition to seismic signal classification and proposed special-purpose VLSI architectures for the implementation. Our studies concentrate on the waveforms where shape information is not important or useful, like seismic signals. EEG and speech signals have similar characteristic as seismic signal. Chapter I defines the problem of study, outlines the approach to the problem and gives relevent literature survey. Chapter II discusses string similarity (distance) measures and recognition procedures. String distances have been classified into two categories: general string distances which are based on the concept of insertion, deletion and substitution transformations and special string distances. General string distances are further classified into a hierarchy of four levels. Symmetric property of string distance has also been discussed. Recognition can be carried out by either nearest-neighbor decision rule or error-correcting parsing. We use a modified Earley's parsing algorithm which does not require an expanded grammar and is able to use symmetric distance.

Chapter III demonstrates the experimental results of seismic discrimination and damage assesment. If shape is not the major feature, pattern segmentation is often simpler. We only need to consider fixed-length segmentation. When shape information is the dominant feature, pattern segmentation is usually associated with primitive recognition. Generally speaking, a fixed-length segmentation is easier to perform; a variable-length segmentation is more efficient in representation. However, a variable-length segmentation usually takes more time in determining the optimal boundary. Furthermore, a variable-length segmentation sometimes starts from fixed-length segmentation and then merges or splits based on a preset criterion. In general, we are in favor of fixed-length segmentation provided a proper length can be easily selected. Feature selection is problem-dependent; therefore we did not emphasize on this subject. Primitive recognition is our first major topic in practical applications. Without any knowledge about the data, we use a clustering procedure to find the optimal number of clusters. Two criteria, increment of merge distance and pseudo F-statistic (PFS), have been used to select cluster number and they show identical results. Finite-state grammars are inferred from the training patterns using the k-tail inference algorithm. Unless the patterns are really generated by a finite-state grammar, chosing small values of $k$ usually worsens the classification result. Our experiments show that uneven merge of states makes the inferred grammar performing poorly in recognition. When the inferred grammar is the canonical grammar, the recognition results of using NN rule and ECP are the same. According to our experiment, the NN rule takes however much less computer time than ECP. A modified dynamic time-warping

system has been used to measure the distance between the seismic waveforms of the building during a strong earthquake. This measurement can be used for damage assesment.

Chapter IV introduces an attributed grammar and parsing for signal recognition in general, and seismic recognition in particular. If we use a canonical grammar as the pattern grammar, it usually contains a large number of production rules and nonterminal symbols. Using attributes will increase the descriptive power of the grammar as well as simplify the syntactic rules of the grammar. We use a length attribute for seismic grammar, which reduces more than 90% of the number of productions and nonterminals from the nonattributed grammar. Attributed seismic grammars also increase the recognition speed while maintaining the same recognition accuracy.

Chapter V contains VLSI architectures for string matching, primitive recognition and feature extraction. Although some special-purpose chips have been developed for signal recognition, for example, spoken word recognition, we are making our systems as general as possible. This is to say our string matcher and primitive recognizer with the exception of feature extractor can be applied to any other pattern recognition problem. They employ parallel processing and pipeline data flow so that very fast throughput can be achieved. This improvement of speed makes real-time pattern recognition possible.

## 6.2 Conclusions

Syntactic pattern recognition has been pointed out as a promising approach to seismic classification (Chen, 1978). While quite a few statistical approaches have been proposed, we are the first to apply syntactic approaches to this problem. With two simple features, our approaches attain better results (91% correct rate) than most of the existing statistical approaches (Tjostheim, 1975; Sarna and Stark, 1980). Our approaches also differ from the syntactic methods in Chapter I in the treatment of primitive selection and grammar construction. A clustering procedure along with two decision criteria constitute the primitive selection algorithm in our approach, while heuristic approaches were used by others, e.g., in Stockman, et al., (1976). Our pattern grammars are inferred from training samples, but most pattern grammars for signal analysis are constructed manually. An attributed grammar for the seismic application is proposed, which could significantly reduce the grammar size and increase the recognition speed. Finally, VLSI architectures are proposed for seismic classification, which include feature extraction, primitive recognition and string matching using (weighted) Levenshtein distance. Our string matcher is different from many contemparory implementations, i.e., exact matching (e.g., in Foster and Kung, 1980), which are not suitable for pattern recognition applications because of the noise and other problems, for example, segmentation and primitive recognition errors; the detail is discussed in chapter V. The computational results can be produced at a constant rate, i.e., constant time complexity, when using our VLSI architectures with pipelined data flow. Although these VLSI systems are developed for seismic classification, they can be applied to

other similar applications.

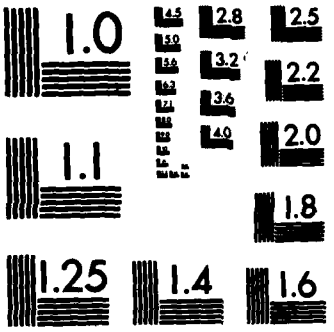
## 6.3 Recommendations

Future works about syntactic seismic signal recognition can be divided into two parts, one is algorithm development, the other is high-speed implementation. (This can also be applied to other signal recognition problems.) In algorithm development, the possibility of using variable-length segmentation should be explored. Stochastic grammars and parsing should be applied when probabilistic information is available. The inclusion of semantic information in pattern primitive is another approach (Tsai and Fu, 1980). A conventional pattern representation contains only syntactic symbols. A typical speech pattern for dynamic time warping contains only numerical infomation. A combination of these two will have both syntactic and semantic information. The distance computation and parsing of such patterns can be separated into syntactic deformation and semantic deformation, and different weights can be assigned to these two deformations. Feature extraction also needs further studies; linear predictive coefficients and features from power spectrum are good candidates.

After the algorithms are developed, they can often be implemented on a parallel architecture, particularly on VLSI architectures. In our string matcher, a global path constraint can be imposed, therefore reduce the number of processors. Those special-purpose chips can be arranged in such a way that the output of one chip is used directly as the input of another chip. Of course, this can happen only wher all the chips have the same processing speed; otherwise, buffers or latches are

AD-A124 398 A SYNTACTIC APPROACH AND VLSI ARCHITECTURES FOR SEISMIC 3/3
SIGNAL CLASSIFICATION(U) PURDUE UNIV LAFAYETTE IN
SCHOOL OF ELECTRICAL ENGINEERING H LIU ET AL. JAN 83
UNCLASSIFIED N00014-79-C-0574 F/G 8/11 NL

END
3-83



required between the chips. Although these chips are for special purposes, flexibility should also be considered. The more flexible the chips are, the more applications they have; therefore, makes their manufacturing cheaper. This combination of algorithm development and technology advance will make many pattern recognition applications practical in both cost and speed.

The application of attributed grammar using length attribute to speech recognition should also be investigated. Suppose two strings $x=aaaaaabbbccc$ and $y=aaaabbcc$ represent different utterances of the same word. If we use string matching and NNR for classification, then $d(x,y) \neq 0$ regardless that we use the conventional Levenshtein distance or weighted Levenshtein distance. Ackroyd (1980) suggested a modified WLD which is computed by subtracting $|I-J|d_{ID}$ from the WLD, where $I$, $J$ are the lengths of the two strings respectively and $d_{ID}$ is the weigth for insertion and deletion. Although this modified WLD can make $d(x,y) = 0$, it will cause other problems, for example, $d(y,z) = 0$ for $z=aaa$. The type 3 WLD proposed in Chapter 2 can solve this problem by letting $D(a,a) = I(a,a) = 0$ for all $a \in \Sigma$. However, there exists one drawback, i.e., there is no restriction on the number of insertions or deletions. An attributed grammar using length attribute can be used to solve this problems without side effects. For example, if string $x$ is the training sample, then the attributed grammar has production $S \rightarrow ABC$ with inherited length attribute ({6,4}, {3,2}, {3,2}) for $(A,B,C)$. This attributed grammar will accept both string $x$ and $y$, but not $z$.

## LIST OF REFERENCES


[1] Ackland, B., Weste, N. and Burr, D.J., 1981. "An integrated multiprocessing array for time warping pattern matching," *Proc. 8th Annu. Symp. on Comput. Archit.* May 12-14, Minneapolis, pp. 197-215.

[2] Ackroyd, M. H., 1980. "Isolated word recognition using the weighted Levenshtein distance," *IEEE Trans. Acous. Speech, Signal Processing*, vol. ASSP-28, no. 2, pp. 243-244.

[3] Aho, A.V. and Peterson, T.G., 1972. "A minimum distance error-correcting parser for context-free languages," *SIAM J. Comput.*, vol. 1, no. 4, pp. 305-312.

[4] Aho, A.V. and Ullman, J.D., 1972. *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, Inc., vol. 1, 542 pp.

[5] Albus, J.E., 1977. "Electrocardiogram interpretation using a stochastic finite state model," in *Syntactic Pattern Recognition Application*, ed. by Fu, K.S., Springer-Verlag, pp. 51-64.

[6] Allen, R.V., 1978. "Automatic earthquake recognition and timing from single traces," *Bull. Seismol. Soc. Amer.*, vol. 6P no. 5, pp. 1521-1532.

[7] Anderson, K.R., 1978. "Automatic Analysis of Microearthquake Network Data," *Geoexplor*, vol. 16, pp. 159-175.

[8] Atal, B.S. and Hanauer, S.L., 1971. "Speech analysis and synthesis by linear prediction of the speech wave," *J. Acoust. Soc. Am.*, vol. 50, no. 2, pp. 637-655.

[9] Bath, M., 1977. *Spectral Analysis in Geophysics*, Elsevier, New York, 563 pp.

[10] Bath, M., 1979. *Introduction to Seismology*, Birkhauser Berlag, Boston, 428 pp.

[11] Biermann, A.W. and Feldman, J.A., 1972. "On the synthesis of finite-state machine," *IEEE Trans. Comput.*, vol. C-21, pp. 592-597.

[12] Blair, C. R., 1960. "A program for correcting spelling errors," *Inform. Contr.*, vol. 3, pp. 60-67.

[13] Bois, P., 1981. "Reservior recognition in petroleum prospection considered as an application of close man-machine communication," *Proc. 2nd Int. Symp. on Comput. Aided Seismic Analysis and Discrimination*, North Dartmouth, Massachusetts, pp. 42-47.

[14] Bolt, B.A., 1976. *Nuclear Explosions and Earthquakes, The Parted Veil.*, W. H. Freeman and Co., San Francisco, 309 pp.

[15] Booth, T.L., 1967. *Sequential Machines and Automata Theory*, Wiley, New York.

[16] Bowen, B. A. and Brown, W. R., 1982. *VLSI Systems Design For Digital Signal Processing*. Prentice-Hall, Inc., New Jersey.

[17] Box, G.E.P. and Jenkins, G.M., 1976. *Time Series Analysis - forecasting and control*, Holden-Day Inc., San Francisco, 575 pp.

[18] Carnahan, B., Luther, H. A. and Wilkes, J. O., 1969. *Applied Numerical Methods*, John Wiley & Sons, Inc., New York.

[19] Chen, C.H., 1978. "Seismic Pattern Recognition," *Geoexplor*, vol. 16, no. 1/2, pp. 133-146.

[20] Chiang, Y. and Fu, K. S., 1981. "Parallel processing for distance computation in syntactic pattern recognition," *Proc. IEEE Workshop on CAPAIDM*. Nov. 11-13, Hot Springs, VA.

[21] Chiang, Y. and Fu, K. S., 1982. "A VLSI architecture for fast context-free language recognition (Earley's algorithm)," *3rd Int. Conf. Distributed Computing Systems*. Oct. 12-15, Ft. Lauderdale, FL.

[22] Chou, S.M. and Fu, K.S., 1975. "Transition networks for pattern recognition," *Tech. Rept.*, TR-EE 75-39, Purdue Universiey, Indiana.

[23] Crespi-Reghizzi, S., 1971. "An effective model for grammar inference," *Proc. IFIP Congress*, Aug., Yugoslavia, pp. 524-529.

[24] Crespi-Reghizzi, S., 1971. "Reduction of enumeration in grammar acquisition," *Proc. 2nd Int. Joint conf. Artif. Intel.*, Sept. 1-3, London, England, pp. 564-552.

[25] Dahlman, O. and Israelson, H., 1977. *Monitoring Underground Nuclear Explosions*, Elsevier Scientific Publishing Co., Amsterdam, the Netherlands.

[26] DeMori, R., 1972. "A descriptive technique for automatic speech recognition," *IEEE Trans. Audio Electroacoustic*, AU-21, pp.89-100.

[27] DeMori, R., 1977. "Syntactic recognition of speech patterns," in *Syntactic Pattern Recognition Application*, ed. by Fu, K.S., Springer-Verlag, New York, pp. 65-94.

[28] Duda, R.D. and Hart, P.E., 1973. *Pattern Classification and Scene Analysis*, Wiley, New York, 482 pp.

[29] Earley, J., 1970. "An efficient context-free parsing algorithm," *CACM* vol. 13, pp. 94-102.

[30] Ehrich, R.W. and Foith, J.P., 1976. "Representation of random waveforms by relational trees," *IEEE Trans. Comput.*, vol. C-25, no. 7, pp.725-736.

[31] Flanagan, J.L., 1972. *Speech Analysis, Synthesis and Perception*, 2nd Ed., Springer-Verlag, New York.

[32] Foster, M. J. and Kung, H. T., 1980. "The desssign of special-purpose VLSI chips," *Computer*, vol. 13, no. 1, pp. 26-40.

[33] Fu, K.S., 1973. "Stochastic languages for picture analysis," *Computer Graphics and Image Processing*, vol. 2, pp. 433-453.

[34] Fu, K.S., 1977. "Error-correcting parsing for syntactic pattern recognition," in *Data Structure, Computer Graphics and Pattern Recognition*, ed. by Klinger, A., et al., Acdemic Press, pp. 449-492.

[35] Fu, K.S., ed., 1977. *Syntactic Pattern Recognition Application*, Springer-Verlag, New York.

[36] Fu, K.S., 1978. "Syntactic pattern recognition and its application to signal processing," *Proceedings of the NATO Advanced Study Institute on Pattern Recognition and Signal Processing*, Series E, Applied Science, no. 29, Sijthoff & Noordhoff International Publishers, The Netherland.

[37] Fu, K.S., 1982. *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Inc., New Jersey, 596 pp.

[38] Fu, K.S. and Huang, T., 1972. "Stochastic grammars and languages," *Intern. Journal of Comput. and Inform. Sci.*, vol. 1, no. 2, pp. 135-170.

[39] Fu, K.S. and Booth, T.L., 1975. "Grammatical inference- introduction and survey," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-5, no 1, pp. 95-111, no. 4, pp. 409-423.

[40] Fu, K.S. and Lu, S.Y., 1977. "A clustering procedure for syntactic patterns," *IEEE Trans. Sys., Man and Cybern*, vol. SMC-7, no. 10, pp. 734-742.

[41] Fu, K.S. and Yao, J.T.P., 1979. "Pattern recognition and damage assesment," The *ASCE EMD Specialty Conference* Austin, Texas.

[42] Fukunaga, K., 1972. *Introduction to Statistical Pattern Recognition*, Academic Press, New York.

[43] Fukunaga, K. and Koontz, W.L.G., 1970. "A criterion and algorithm for grouping data," *IEEE Trans. Comput.*, vol. C-19, pp. 917-923.

[44] Fung, L.W. and Fu, K.S., 1975. "Stochastic syntactic decoding for pattern classification," *IEEE Trans. Comput.*, vol. C-24, pp. 662-669.

[45] Giese, D.A., Bourne, J.R. and Ward, J.W., 1979. "Syntactic analysis of the electroencephalogram," *IEEE Tran. Syst., Man., Cybern.*, vol.9, no.8, pp.429-435.

[46] Guibas, L. J., Kung, H. T. and Thompson, C. D., 1979. "Direct VLSI implementation of combinatorial algorithms," *Caltech Conf. on VLSI*, Jan., pp. 509-525.

[47] Hall, P.A.V. and Dowling, G.R., 1980. "Approximate string matching," *ACM Comput. Surveys*, vol. 12, no. 4, pp. 381-402.

[48] Horowitz, S.L., 1975. "A syntactic algorithm for peak detection in waveforms with applications to cardiography," *CACM*, vol. 18, no. 5, pp. 281-285.

[49] Horowitz, S.L., 1977. "Peak recognition in waveforms," in *Syntactic Pattern Recognition Application*, ed. by Fu, K.S., Springer-Verlag, pp. 31-49.

[50] Hwang, K. and Cheng, Y. H., 1981. "Partioned matrix algcrithms and VLSI structures for large scale matrix computations," *5th Symp. Comput. Arithmetic*, May 18-19, Ann Arbor, Mich., pp. 222-232.

[51] Ishizuka, M., Fu., K.S. and Yao, J.T.P., 1981. "Inexact inference for rule-based damage assesment of existing structure," *Tech. Rep.*, CE-STR-81-5, Purdue University, Indiana.

[52] Joshi, A.K., 1973. "Remarks on some aspects of language structure and their relevance to pattern analysis," *Pattern Recognition*, vol. 5, no. 4.

[53] Knuth, D.E., 1968. "Semantics of context-free languages," *J. Math. Sys. Theory*, vol. 2, pp. 127-146.

[54] Kuck, D. J., 1978. *The Structure of Computers and Computations*. John-Wiley and Sons, New York.

[55] Kulkarni, A. V. and Yen, D. W. L., 1982. "Systolic processing and an implementation for signal and image processing," *IEEE Trans. Comput.*, vol. C-31, no. 10, pp. 1000-1009.

[56] Kung, H.T., 1979. "Let's design algorithm for VLSI systems," *Proc. Caltech Conf. on VLSI*, Jan., Pasadena, CA, pp. 65-90.

[57] Kung, H. T., 1982. "Why systolic architectures ?" *Computer*, vol. 15, no. 1, pp. 37-46.

[58] Lee, H.C. and Fu, K.S., 1972. "A stochastic syntax analysis procedure and its application to pattern classification," *IEEE Trans. Comput.*, vol. C-21, pp. 660-666.

[59] Lee, H.C. and Fu, K.S., 1972. "A syntactic pattern recognition system with learning capability," *Proc. Int. Symp. Comput. and Inform. Sci.* Dec. 14-16, Miami Beach, Florida.

[60] Lee, H.C. and Fu, K.S., 1972. "Stochastic linguistics for pattern recognition," *Tech. Rep.*, TR-EE 72-17, Purdue Universiry, Lafayette, Indiana.

[61] Levenshtein, V.I., 1966. "Binary codes capable of correcting deletions, insertions, and reversals," *Sov. Phys. Dokl.*, vol. 10, pp. 707-710.

[62] Lozano-Peiez, T., 1977. "Parsing Intensity Profile," *Computer Graphics and Image Processing*, vol. 6, pp. 43-60.

[63] Lu, S.Y. and Fu, K.S., 1977. "Stochastic error-correcting syntax analysis for recognition of noisy pattern," *IEEE Trans. Comput.*, vol. C-26, no. 12., pp.1268-1276.

[64] Lu, S.Y. and Fu, K.S., 1978a. "A syntactic approach to texture analysis," *Computer Graphics and Image Processing*, vol. 7, no. 3, pp. 303-330.

[65] Lu, S.Y. and Fu, K.S., 1978b. "Error-correcting tree automata for syntactic pattern recognition," *IEEE Trans. Comput.*, vol. C-27, Nov., pp. 1040-1053.

[66] Lu, S.Y. and Fu, K.S., 1979. "Stochastic tree grammar inference for texture analysis and discrimination," *Computer Graphics and Image Processing*, vol. 9, pp. 234-245.

[67] Lyon, G., 1974. "Syntax-directed least-error analysis for context-free languages: a practical approach," *CACM*, vol. 17, no. 1 pp. 3-14.

[68] Mead, C.A. and Conway, L.A., 1980. *Introduction to VLSI Systems*, Addison-Wesley, 396 pp.

[69] Miclet, L., 1980. "Regular inference with a tail-clustering method," *IEEE Trans. Syst., Man., Cybern.*, vol. SMC-10, pp. 737-743.

[70] Mottl', V.V. and Muchnik, I.B., 1979. "Linguistic analysis of experimental curves," *Proc. IEEE*, vol. 67, no. 5, pp. 714-736.

[71] Mukhopadhyay, A., 1979. "Hardware algorithm for nonnumeric computation," *IEEE Trans. Comput.*, vol. C-28, no. 6, pp.384-394.

[72] Okuda, T, Tanaka, E and Kasai, T, 1976. "A method for the correction of garbled words based on the Levenshtein metric," *IEEE Trans. Comput.*, vol. C-25, pp. 172-178.

[73] Oppenheim, A.V., 1970. "Speech spectrograms using the fast fourier transform," *IEEE Spectrum*, vol. 7, pp. 57-62.

[74] Oppenheim, A.V. and Schafer, R.W., 1975. *Digital Signal Processing*, Prentice-Hall Inc., Englewood Cliffs, New Jersey.

[75] Pavlidis, T., 1971. "Linguistic analysis of waveforms," in *Software Engineering*, ed. by Tou, J. T., vol. 2, Acdemic, New Yor, pp.203-225.

[76] Pavlidis, T., 1973. "Waveform segmentation through functional approximation," *IEEE Trans. Comput.*, vol. C-22, pp.689-697.

[77] Pavlidis, T. and Horowitz, S.L., 1974. "Segmentation of plane curves," *IEEE Trans. Comput.*, vol. C-23, pp.860-870.

[78] Reddy, D.R., ed., 1975. *Speech Recognition*, Acdemic Press, New York.

[79] Sakoe, H. and Chiba, S., 1978. "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 43-49.

[80] Sandvin, O. and Tjostheim, D., 1978. "Multivariate autoregressive representation of seismic P-wave signals with application to short-period discrimination," *Bull. Seism. Soc. Am.*, vol. 68, pp. 735-756.

[81] Sankar, P.V. and Rosenfeld, A., 1979. "Hierarchical representation of waveforms," *IEEE Trans. Patt. Analy. and Mach. Intel.*, vol. PAMI-1, no. 1, pp. 73-79.

[82] Sarna C.S. and Stark H., 1980. "Pattern recognition of waveforms using modern spectral estimation techniques and its application to earthquake / explosion data," *Proc. 5th Intl. Conf. on Pattern Recognition*, Dec. 1-4, Miami Beach, FL.

[83] Schafer, R.W. and Rabiner, L.R., 1975. "Digital representation of speech signal," *Proc. IEEE*, vol. 63, pp. 662-677.

[84] Stewart, S.W., 1977. "Real-time detection and location of local seismic events in central California," *Bull. Seism. Soc. Am.*, vol. 67, pp. 433-452.

[85] Stockman, G., Kanel, L.N. and Kyle, M.C., 1976. "Structural pattern recognition of carotid pulse waves using a general waveform parsing system," *CACM*, vol. 19, no. 12, pp. 688-695.

[86] Stone, H. S., ed., 1980. *Introduction to Computer Architecture*. Science Research Associates, Inc., Chicago.

[87] Swerling, S. 1982. "Computer-aided engineering," *IEEE Spectrum*, vol. 19, no. 11, pp. 37-41.

[88] Tanaka, E. and Kasai, T., 1972. "A correcting method of garbled languages using ordered key letters," *Trans Inst. Elec. Commun. Eng. (Japan)*, vol. 55-D, pp. 363-370.

[89] Tai, J.W. and Fu, K.S., 1981. "Semantic syntax-directed translation for ipctorial pattern recognition," *Tech. Rep.*, TR-EE 81-38, Purdue University.

[90] Tai, J. W. and Fu, K. S., 1982. "Inference of a class of CFPG by means of semantic rules," *Int'l J. of Comput. and Inf. Sci.*, vol. 11, no. 1, pp. 1-23.

[91] Tang. G.Y. and Huang, T.S., 1979. "A syntactic-semantic approach to image understanding and creation," *IEEE Trans. Patt. Anal. Mach. Intel.*, vol. PAMI-1.

[92] Thomason, M.G. and Gonzalez, R.C., 1975. "Error detection and classification in syntactic pattern structures," *IEEE Trans. Comput.*, vol. C-24.

[93] Thomason, M.G. and Gonzalez, R.C., 1975. "Syntactic recognition of imperfectly specified patterns," *IEEE Trans. Comput.*, vol. C-24, no. 1, pp. 93-95.

[94] Tjostheim, D., 1975. "Autoregressive representation of seismic P-wave signals with an application to the problem of short-period discriminants," *Geophys. J. R. Astr. Soc.*, vol. 43, pp. 269-291.

[95] Tjostheim, D., 1977. "Recognition of waveforms using autoregressive feature extraction," *IEEE Trans. Comput.*, vol. C-26, pp. 268-270.

[96] Tjostheim, D., 1978. "Improved seismic discrimination using pattern recognition," *Phys. Earth Planet, Inter.*, vol. 16, pp. 85-108.

[97] Tjostheim, D. and Sandvin, O., 1979. "Multivariate Autoregressive Feature Extraction and the Recognition of Multichannel Waveforms," *IEEE Trans. Patt. Analys. and Mach. Intell.*, vol. PAMI-1, no. 1, pp.80-86.

[98] Tomek, I., 1975. "More on Piecewise Linear Approximation," *Comput. Biomed. Res.*, vol. 8, pp.568-572.

[99] Tsai, W.H. and Fu, K.S., 1979. "A pattern deformation model and Bayes error-correcting recognition system," *IEEE Trans. Sys. Man and Cyber.*, vol. SMC-9, no. 12,pp. 745-756.

[100] Tsai, W.H. and Fu, K.S., 1980. "Attributed grammar-a tool for combing syntactic and statistical approach to pattern recognition," *IEEE Trans. Sys. Man and Cyber.*, vol. SMC-10, no. 12, pp. 873-885.

[101] Vogel, M.A. and Wong, A.K.C., 1978. "PFS Clustering Method," *IEEE Trans. Patt. Analy. Mach. Intel.*, vol. PAMI-1, no. 3, pp. 237-245.

[102] Wagner, R.A., 1974. "Order-n correction of regular languages," *CACM*, vol. 17, no. 5, pp. 265-268.

[103] Wagner, R.A. and Fischer, M.J., 1974. "The string-to-string correction problem," *J. ACM* vol. 21, no. 1, pp. 168-178.

[104] Yao, J.T.P., 1979. "Damage assessment and reliability evaluation of existing structures," *Eng. Struct.*, vol. 1, pp. 245-251.

[105] You, K.C., 1978. "Syntactic shape recognition using attributed grammars," *Tech. Rep.*, TR-EE 78-38, Purdue University, Indiana.

[106] You, K.C. and Fu, K.S., 1979. "A syntactic approach to shape recognition using attributed grammar," *IEEE Trans. Sys. Man and Cyber.*, vol. SMC-9, no. 6, pp. 334-345.

# APPENDIX A

## FLOW CHARTS FOR THE SIMULATIONS

Appendix A gives the flow charts for the simulations in Section 5.5. Figure A.1 is the flow chart for feature extraction, Figure A.2 is the flow chart for primitive recognition, and Figure A.3 is the flow chart for string matching.

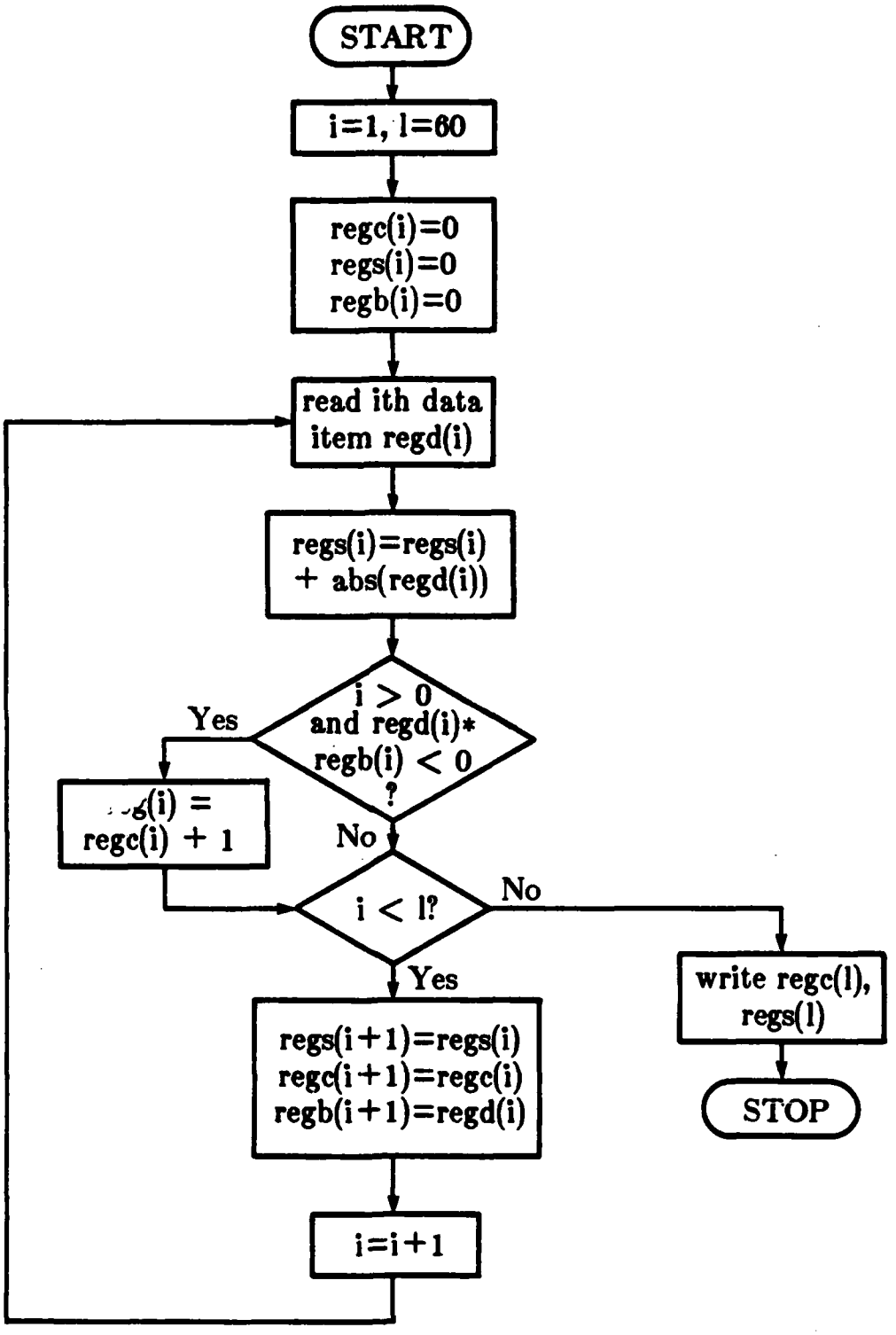



Figure A.1 Flow chart for the simulation of feature extraction.

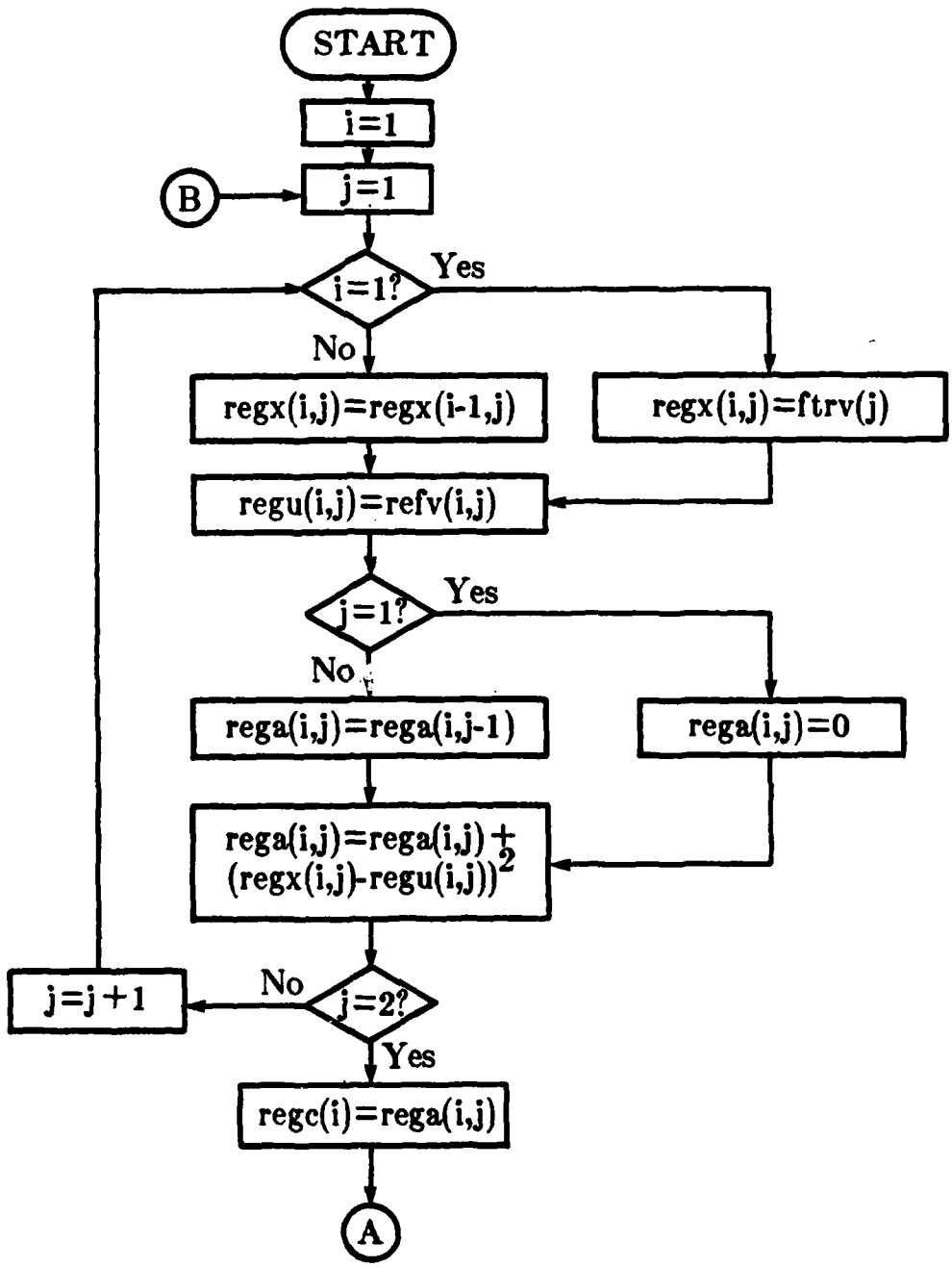



Figure A.2 Flow chart for the simulation of primitive recognition.

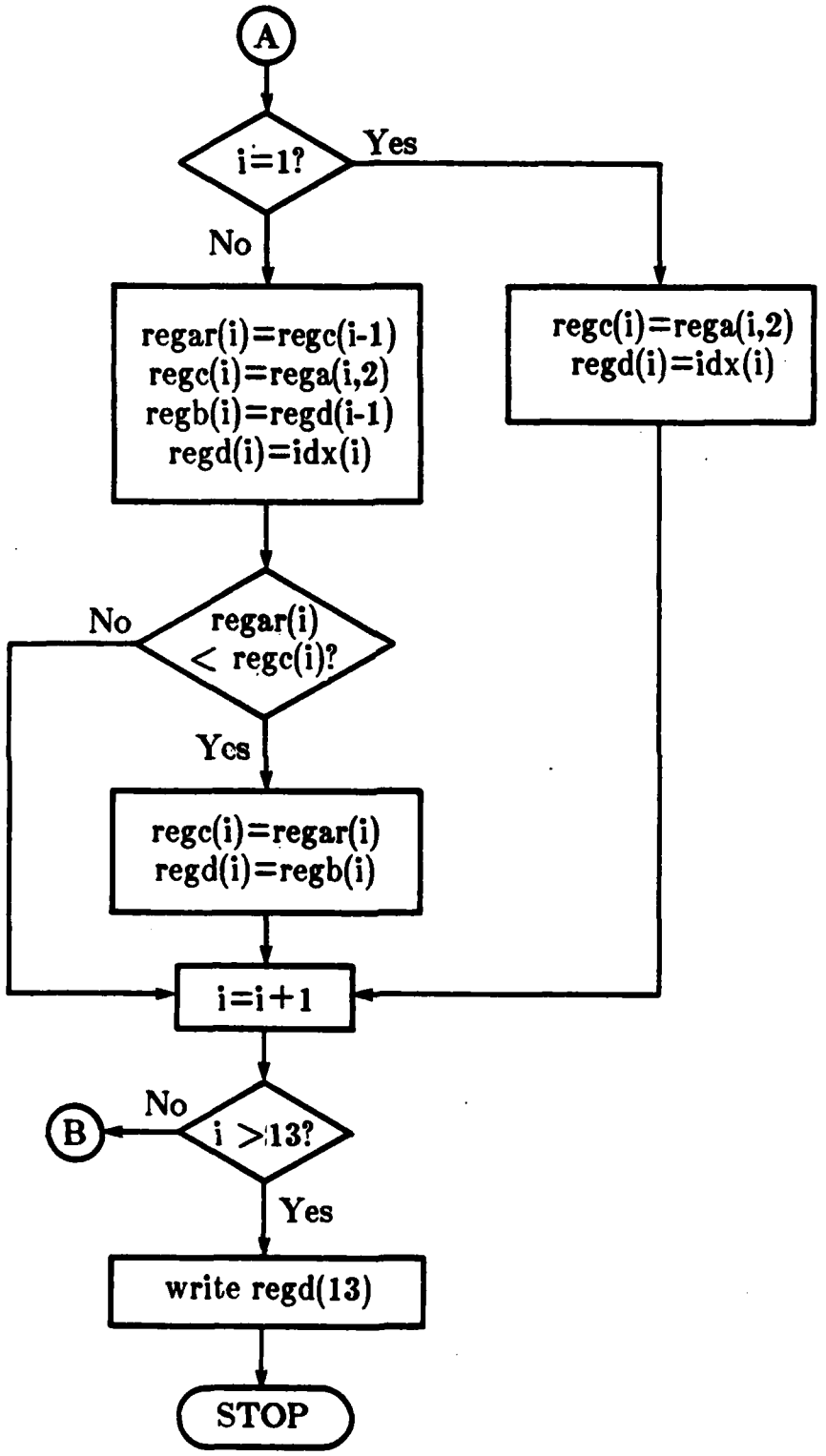



Figure A.2 (Continued)

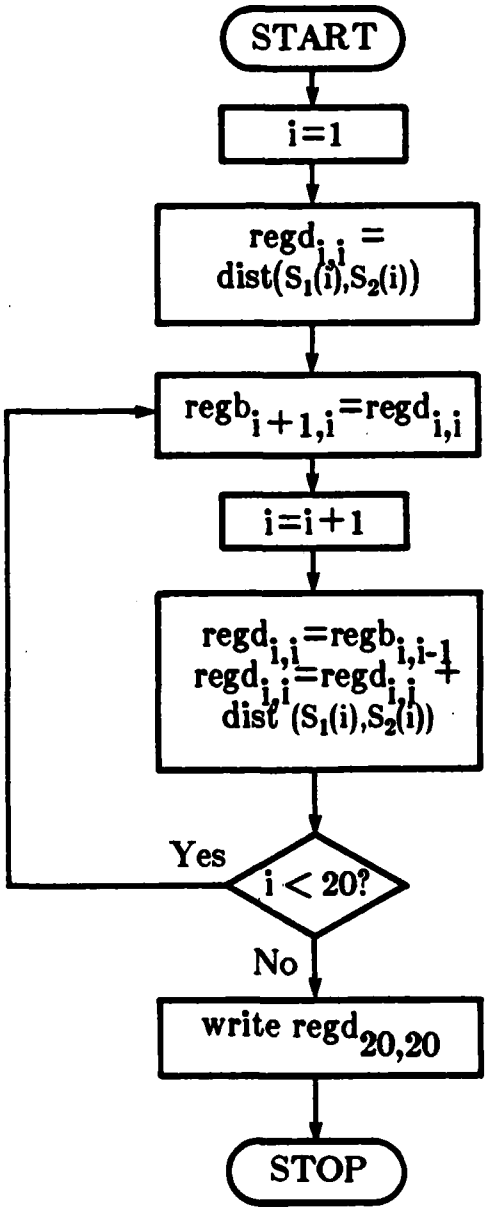



Figure A.3 Flow chart for the simulation of string matching.

# APPENDIX B

## STEP-BY-STEP SIMULATION RESULTS

Table B.1 shows the intermediate results of feature extraction at each time interval for the seismic signal shown in Figure B.1. The symbols $a, b, S, c, x, y$ and $d$ are described in Figure 5.4(b). We use a linearly-connected array of 60 processors. Therefore, for a specific seismic segment, it takes 60 unit times to pass through the processor array. Since the data can be pipelined, it takes only one unit time to extract the feature from each segment. The inputs to Table B.2 are the outputs from Table B.1 after normalization. Table B.2 shows the intermediate results of primitive recognition at each time interval. At time $n$ and $2n$, $n$ = 1, 2, ..., 13, the simulation executes 'compute' operation. At time $3n$, $n$ = 1, 2, ..., 13, the simulation executes 'compare' operation. The symbols $a, x, u, b, y$ and $v$ of 'compute' operation are described in Figure 5.7(a). The symbols $a, b, c, d, y$ and $z$ of 'compare' operation are described in Figure 5.7(b). A specific feature vector takes 39 unit times to pass through the processor array. Since the feature vectors can be pipelined, it takes two unit times to assign a primitive to each feature vector. The output '$g$' from Table B.2 is the $4th$ symbol of the second string in Table B.3. Table B.3 shows the intermediate results of string matching using the weighted Levenstein distance. Since only substitution errors are considered, the computation is straightforward. The symbols $x, y, d$ and $b$ represent the registers as

described in Figure 5.16. A specific pair of strings take 39 unit times to pass through the processor array. Since the strings can be pipelined, it takes 3 unit times to compute the distance between an unknown and a reference string. The recognition results will not be known until we compare against all the (100) reference strings.

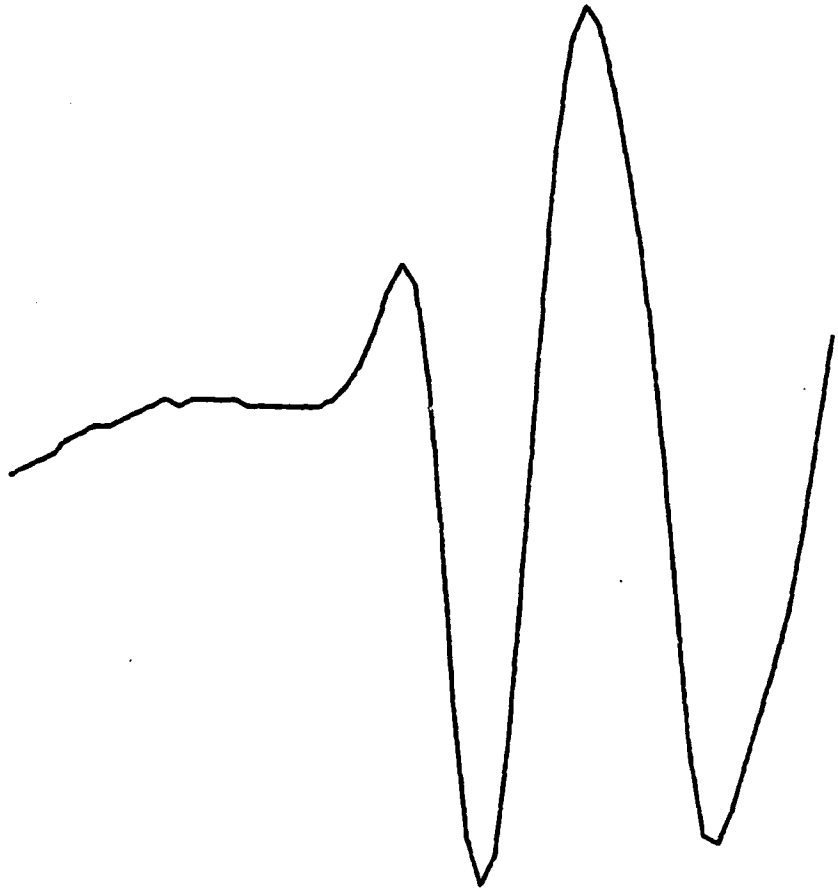


Figure B.1 Seismic segment (60 points) used in the simulation to generate intermediate results of Table B.1 and Table B.2.

TABLE B.1

**The intermediate results of feature extraction at each time interval for the seismic segment shown in Figure B.1**

| | | | | |
|---|---|---|---|---|
| t = 1 | a = -0.732877 | b = 0.000000 | S = 0.732877 | c = 0 |
| | x = -0.732877 | y = 0.732877 | d = 0 | |
| t = 2 | a = -0.732877 | b = -0.732877 | S = 1.465755 | c = 0 |
| | x = -0.732877 | y = 1.465755 | d = 0 | |
| t = 3 | a = -0.732875 | b = -0.732877 | S = 2.198629 | c = 0 |
| | x = -0.732875 | y = 2.198629 | d = 0 | |
| t = 4 | a = -0.578423 | b = -0.732875 | S = 2.777052 | c = 0 |
| | x = -0.578423 | y = 2.777052 | d = 0 | |
| t = 5 | a = -0.423971 | b = -0.578423 | S = 3.201023 | c = 0 |
| | x = -0.423971 | y = 3.201023 | d = 0 | |
| t = 6 | a = -0.115067 | b = -0.423971 | S = 3.316090 | c = 0 |
| | x = -0.115067 | y = 3.316090 | d = 0 | |
| t = 7 | a = 0.039385 | b = -0.115067 | S = 3.355475 | c = 0 |
| | x = 0.039385 | y = 3.355475 | d = 1 | |
| t = 8 | a = 0.193837 | b = 0.039385 | S = 3.549313 | c = 1 |
| | x = 0.193837 | y = 3.549313 | d = 1 | |
| t = 9 | a = 0.193837 | b = 0.193837 | S = 3.743150 | c = 1 |
| | x = 0.193837 | y = 3.743150 | d = 1 | |
| t = 10 | a = 0.348289 | b = 0.193837 | S = 4.091439 | c = 1 |
| | x = 0.348289 | y = 4.091439 | d = 1 | |
| t = 11 | a = 0.502741 | b = 0.348289 | S = 4.594181 | c = 1 |
| | x = 0.502741 | y = 4.594181 | d = 1 | |

| | | | | |
|---|---|---|---|---|
| t = 12 | a = 0.657193 | b = 0.502741 | S = 5.251374 | c = 1 |
| | x = 0.657193 | y = 5.251374 | d = 1 | |
| t = 13 | a = 0.811646 | b = 0.657193 | S = 6.063020 | c = 1 |
| | x = 0.811646 | y = 6.063020 | d = 1 | |
| t = 14 | a = 0.657193 | b = 0.811646 | S = 6.720213 | c = 1 |
| | x = 0.657193 | y = 6.720213 | d = 1 | |
| t = 15 | a = 0.811646 | b = 0.657193 | S = 7.531859 | c = 1 |
| | x = 0.811646 | y = 7.531859 | d = 1 | |
| t = 16 | a = 0.811646 | b = 0.811646 | S = 8.343505 | c = 1 |
| | x = 0.811646 | y = 8.343505 | d = 1 | |
| t = 17 | a = 0.811646 | b = 0.811646 | S = 9.155150 | c = 1 |
| | x = 0.811646 | y = 9.155150 | d = 1 | |
| t = 18 | a = 0.811646 | b = 0.811646 | S = 9.966796 | c = 1 |
| | x = 0.811646 | y = 9.966796 | d = 1 | |
| t = 19 | a = 0.657193 | b = 0.811646 | S = 10.623989 | c = 1 |
| | x = 0.657193 | y = 10.623989 | d = 1 | |
| t = 20 | a = 0.657193 | b = 0.657193 | S = 11.281182 | c = 1 |
| | x = 0.657193 | y = 11.281182 | d = 1 | |
| t = 21 | a = 0.657193 | b = 0.657193 | S = 11.938375 | c = 1 |
| | x = 0.657193 | y = 11.938375 | d = 1 | |
| t = 22 | a = 0.657193 | b = 0.657193 | S = 12.595569 | c = 1 |
| | x = 0.657193 | y = 12.595569 | d = 1 | |
| t = 23 | a = 0.657193 | b = 0.657193 | S = 13.252762 | c = 1 |
| | x = 0.657193 | y = 13.252762 | d = 1 | |
| t = 24 | a = 0.657193 | b = 0.657193 | S = 13.909955 | c = 1 |
| | x = 0.657193 | y = 13.909955 | d = 1 | |
| t = 25 | a = 0.811646 | b = 0.657193 | S = 14.721601 | c = 1 |
| | x = 0.811646 | y = 14.721601 | d = 1 | |
| t = 26 | a = 1.120550 | b = 0.811646 | S = 15.842150 | c = 1 |

| | | | | |
|---|---|---|---|---|
| | x = 1.120550 | y = 15.842150 | d = 1 | |
| t = 27 | a = 1.583906 | b = 1.120550 | S = 17.426056 | c = 1 |
| | x = 1.583906 | y = 17.426056 | d = 1 | |
| t = 28 | a = 2.356166 | b = 1.583906 | S = 19.782223 | c = 1 |
| | x = 2.356166 | y = 19.782223 | d = 1 | |
| t = 29 | a = 3.282878 | b = 2.356166 | S = 23.065102 | c = 1 |
| | x = 3.282878 | y = 23.065102 | d = 1 | |
| t = 30 | a = 3.900687 | b = 3.282878 | S = 26.965788 | c = 1 |
| | x = 3.900687 | y = 26.965788 | d = 1 | |
| t = 31 | a = 3.437330 | b = 3.900687 | S = 30.403118 | c = 1 |
| | x = 3.437330 | y = 30.403118 | d = 1 | |
| t = 32 | a = 1.275002 | b = 3.437330 | S = 31.678120 | c = 1 |
| | x = 1.275002 | y = 31.678120 | d = 1 | |
| t = 33 | a = -2.122946 | b = 1.275002 | S = 33.801064 | c = 1 |
| | x = -2.122946 | y = 33.801064 | d = 2 | |
| t = 34 | a = -5.984246 | b = -2.122946 | S = 39.785309 | c = 2 |
| | x = -5.984246 | y = 39.785309 | d = 2 | |
| t = 35 | a = -8.918835 | b = -5.984246 | S = 48.704144 | c = 2 |
| | x = -8.918835 | y = 48.704144 | d = 2 | |
| t = 36 | a = -10.000000 | b = -8.918835 | S = 58.704144 | c = 2 |
| | x = -10.000000 | y = 58.704144 | d = 2 | |
| t = 37 | a = -9.227739 | b = -10.000000 | S = 67.931885 | c = 2 |
| | x = -9.227739 | y = 67.931885 | d = 2 | |
| t = 38 | a = -6.447603 | b = -9.227739 | S = 74.379486 | c = 2 |
| | x = -6.447603 | y = 74.379486 | d = 2 | |
| t = 39 | a = -2.122946 | b = -6.447603 | S = 76.502434 | c = 2 |
| | x = -2.122946 | y = 76.502434 | d = 2 | |
| t = 40 | a = 2.201714 | b = -2.122946 | S = 78.704147 | c = 2 |
| | x = 2.201714 | y = 78.704147 | d = 3 | |

| | | | | |
|---|---|---|---|---|
| t = 41 | a = 6.371919 | b = 2.201714 | S = 85.076065 | c = 3 |
| | x = 6.371919 | y = 85.076065 | d = 3 | |
| t = 42 | a = 8.997603 | b = 6.371919 | S = 94.073669 | c = 3 |
| | x = 8.997603 | y = 94.073669 | d = 3 | |
| t = 43 | a = 9.769864 | b = 8.997603 | S = 103.843536 | c = 3 |
| | x = 9.769864 | y = 103.843536 | d = 3 | |
| t = 44 | a = 9.306508 | b = 9.769864 | S = 113.150047 | c = 3 |
| | x = 9.306508 | y = 113.150047 | d = 3 | |
| t = 45 | a = 8.070891 | b = 9.306508 | S = 121.220940 | c = 3 |
| | x = 8.070891 | y = 121.220940 | d = 3 | |
| t = 46 | a = 6.526371 | b = 8.070891 | S = 127.747314 | c = 3 |
| | x = 6.526371 | y = 127.747314 | d = 3 | |
| t = 47 | a = 4.672946 | b = 6.526371 | S = 132.420258 | c = 3 |
| | x = 4.672946 | y = 132.420258 | d = 3 | |
| t = 48 | a = 2.356166 | b = 4.672946 | S = 134.776428 | c = 3 |
| | x = 2.356166 | y = 134.776428 | d = 3 | |
| t = 49 | a = -0.732875 | b = 2.356166 | S = 135.509308 | c = 3 |
| | x = -0.732875 | y = 135.509308 | d = 4 | |
| t = 50 | a = -3.821918 | b = -0.732875 | S = 139.331223 | c = 4 |
| | x = -3.821918 | y = 139.331223 | d = 4 | |
| t = 51 | a = -6.910959 | b = -3.821918 | S = 146.242188 | c = 4 |
| | x = -6.910959 | y = 146.242188 | d = 4 | |
| t = 52 | a = -8.764383 | b = -6.910959 | S = 155.006577 | c = 4 |
| | x = -8.764383 | y = 155.006577 | d = 4 | |
| t = 53 | a = -8.918835 | b = -8.764383 | S = 163.925415 | c = 4 |
| | x = -8.918835 | y = 163.925415 | d = 4 | |
| t = 54 | a = -8.146575 | b = -8.918835 | S = 172.071991 | c = 4 |
| | x = -8.146575 | y = 172.071991 | d = 4 | |

| | | | | |
|---|---|---|---|---|
| t = 55 | a = -7.065411 | b = -8.146575 | S = 179.137405 | c = 4 |
| | x = -7.065411 | y = 179.137405 | d = 4 | |
| t = 56 | a = -5.984246 | b = -7.065411 | S = 185.121658 | c = 4 |
| | x = -5.984246 | y = 185.121658 | d = 4 | |
| t = 57 | a = -4.903082 | b = -5.984246 | S = 190.024734 | c = 4 |
| | x = -4.903082 | y = 190.024734 | d = 4 | |
| t = 58 | a = -3.667466 | b = -4.903082 | S = 193.692200 | c = 4 |
| | x = -3.667466 | y = 193.692200 | d = 4 | |
| t = 59 | a = -1.814041 | b = -3.667466 | S = 195.506241 | c = 4 |
| | x = -1.814041 | y = 195.506241 | d = 4 | |
| t = 60 | a = 0.193837 | b = -1.814041 | S = 195.700073 | c = 4 |
| | x = 0.193837 | y = 195.700073 | d = 5 | |

TABLE B.2

The intermediate results of primitive recognition at each time interval for the feature vector from Table B.1 after normalization.

| | | | | |
|---|---|---|---|---|
| t = 1 | a = .000 | x = -.161 | u = -1.718 | |
| | b = 2.426 | y = -.161 | v = -1.718 | |
| t = 2 | a = 2.426 | x = 19.252 | u = -2.108 | |
| | b = 458.711 | y = 19.252 | v = -2.108 | |
| t = 3 | a = ********** | b = 458.711 | c = '*' | d = 'a' |
| | y = 458.711 | z = 'a' | | |
| t = 4 | a = .000 | x = -.161 | u = 3.337 | |
| | b = 12.232 | y = -.161 | v = 3.337 | |
| t = 5 | a = 12.232 | x = 19.252 | u = -1.740 | |
| | b = 452.920 | y = 19.252 | v = -1.740 | |
| t = 6 | a = 458.711 | b = 452.920 | c = 'a' | d = 'b' |
| | y = 452.920 | z = 'b' | | |
| t = 7 | a = .000 | x = -.161 | u = -.180 | |
| | b = .000 | y = -.161 | v = -.180 | |
| t = 8 | a = .000 | x = 19.252 | u = -2.387 | |
| | b = 468.287 | y = 19.252 | v = -2.387 | |
| t = 9 | a = 452.920 | b = 468.287 | c = 'b' | d = 'c' |
| | y = 452.920 | z = 'b' | | |
| t = 10 | a = .000 | x = -.161 | u = -1.229 | |
| | b = 1.142 | y = -.161 | v = -1.229 | |
| t = 11 | a = 1.142 | x = 19.252 | u = .987 | |
| | b = 334.762 | y = 19.252 | v = .987 | |

| | | | | |
|---|---|---|---|---|
| t = 12 | a = 452.920 | b = 334.762 | c = 'b' | d = 'd' |
| | y = 334.762 | z = 'd' | | |
| t = 13 | a = .000 | x = -.161 | u = .467 | |
| | b = .394 | y = -.161 | v = .467 | |
| t = 14 | a = .394 | x = 19.252 | u = 1.049 | |
| | b = 331.763 | y = 19.252 | v = 1.049 | |
| t = 15 | a = 334.762 | b = 331.763 | c = 'd' | d = 'e' |
| | y = 331.763 | z = 'e' | | |
| t = 16 | a = .000 | x = -.161 | u = .427 | |
| | b = .345 | y = -.161 | v = .427 | |
| t = 17 | a = .345 | x = 19.252 | u = .114 | |
| | b = 366.632 | y = 19.252 | v = .114 | |
| t = 18 | a = 331.763 | b = 366.632 | c = 'e' | d = 'f' |
| | y = 331.763 | z = 'e' | | |
| t = 19 | a = .000 | x = -.161 | u = -.407 | |
| | b = .061 | y = -.161 | v = -.407 | |
| t = 20 | a = .061 | x = 19.252 | u = 1.284 | |
| | b = 322.939 | y = 19.252 | v = 1.284 | |
| t = 21 | a = 331.763 | b = 322.939 | c = 'e' | d = 'g' |
| | y = 322.939 | z = 'g' | | |
| t = 22 | a = .000 | x = -.161 | u = -.321 | |
| | b = .026 | y = -.161 | v = -.321 | |
| t = 23 | a = .026 | x = 19.252 | u = .440 | |
| | b = 353.928 | y = 19.252 | v = .440 | |
| t = 24 | a = 322.939 | b = 353.928 | c = 'g' | d = 'h' |
| | y = 322.939 | z = 'g' | | |
| t = 25 | a = .000 | x = -.161 | u = 1.431 | |
| | b = 2.533 | y = -.161 | v = 1.431 | |

```
t = 26   a = 2.533       x = 19.252    u = .169
         b = 366.713     y = 19.252    v = .169

t = 27   a = 322.939     b = 366.713   c = 'g'      d = 'i'
         y = 322.939     z = 'g'

t = 28   a = .000        x = -.161     u = -.307
         b = .021        y = -.161     v = -.307

t = 29   a = .021        x = 19.252    u = -.573
         b = 393.089     y = 19.252    v = -.573

t = 30   a = 322.939     b = 393.089   c = 'g'      d = 'j'
         y = 322.939     z = 'g'

t = 31   a = .000        x = -.161     u = 1.486
         b = 2.710       y = -.161     v = 1.486

t = 32   a = 2.710       x = 19.252    u = -.940
         b = 410.457     y = 19.252    v = -.940

t = 33   a = 322.939     b = 410.457   c = 'g'      d = 'k'
         y = 322.939     z = 'g'

t = 34   a = .000        x = -.161     u = -1.414
         b = 1.570       y = -.161     v = -1.414

t = 35   a = 1.570       x = 19.252    u = -.256
         b = 382.141     y = 19.252    v = -.256

t = 36   a = 322.939     b = 382.141   c = 'g'      d = 'l'
         y = 322.939     z = 'g'

t = 37   a = .000        x = -.161     u = .477
         b = .406        y = -.161     v = .477

t = 38   a = .406        x = 19.252    u = -.757
         b = 400.778     y = 19.252    v = -.757

t = 39   a = 322.939     b = 400.778   c = 'g'      d = 'm'
         y = 322.939     z = 'g'
```

TABLE B.3

The intermediate results of string matching at each time interval between strings '*acag hfijjmjfmmkmjjjm*' and '*mklgifdifhffm killibb*'. The output d = 5.742 is the distance between these two strings.

| | | | |
|---|---|---|---|
| t = 1 | x = 'a' | y = 'm' | d = 0.485 |
| t = 2 | x = 'c' | y = 'm' | b = 0.485 |
| t = 3 | x = 'c' | y = 'k' | d = 0.900 |
| t = 4 | x = 'a' | y = 'k' | b = 0.900 |
| t = 5 | x = 'a' | y = 'l' | d = 1.253 |
| t = 6 | x = 'g' | y = 'l' | b = 1.253 |
| t = 7 | x = 'g' | y = 'g' | d = 1.253 |
| t = 8 | x = 'h' | y = 'g' | b = 1.253 |
| t = 9 | x = 'h' | y = 'i' | d = 1.586 |
| t = 10 | x = 'f' | y = 'i' | b = 1.586 |
| t = 11 | x = 'f' | y = 'f' | d = 1.586 |
| t = 12 | x = 'i' | y = 'f' | b = 1.586 |
| t = 13 | x = 'i' | y = 'd' | d = 2.109 |
| t = 14 | x = 'j' | y = 'd' | b = 2.109 |
| t = 15 | x = 'j' | y = 'i' | d = 2.464 |
| t = 16 | x = 'j' | y = 'i' | b = 2.464 |

t = 17 x = 'j' y = 'f' d = 2.654

t = 18 x = 'm' y = 'f' b = 2.654

t = 19 x = 'm' y = 'h' d = 2.924

t = 20 x = 'j' y = 'h' b = 2.924

t = 21 x = 'j' y = 'f' d = 3.113

t = 22 x = 'f' y = 'f' b = 3.113

t = 23 x = 'f' y = 'f' d = 3.113

t = 24 x = 'm' y = 'f' b = 3.113

t = 25 x = 'm' y = 'm' d = 3.113

t = 26 x = 'm' y = 'm' b = 3.113

t = 27 x = 'm' y = 'k' d = 3.306

t = 28 x = 'k' y = 'k' b = 3.306

t = 29 x = 'k' y = 'i' d = 3.515

t = 30 x = 'm' y = 'i' b = 3.515

t = 31 x = 'm' y = 'l' d = 3.882

t = 32 x = 'j' y = 'l' b = 3.882

t = 33 x = 'j' y = 'l' d = 4.099

t = 34 x = 'j' y = 'l' b = 4.099

t = 35 x = 'j' y = 'i' d = 4.454

t = 36 x = 'j' y = 'i' b = 4.454

t = 37 x = 'j' y = 'b' d = 5.173

**t = 39 x = 'm' y = 'b' d = 5.742**


END